Lecture 4a: Adversarial Search

CSCI 360

Introduction to Artificial Intelligence

USC

Here is where we are…

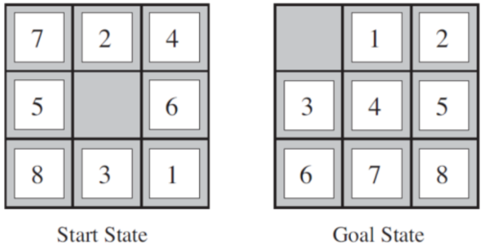
Week	30000D	30282R	Topics	Chapters
1	1/7	1/8	Intelligent Agents	[Ch 1.1-1.4 and 2.1-2.4]
	1/9	1/10	Problem Solving and Search	[Ch 3.1-3.3]
2	1/14	1/15	Uninformed Search	[Ch 3.3-3.4]
	1/16	1/17	Heuristic Search (A*)	[Ch 3.5]
3	1/21	1/22	Heuristic Functions	[Ch 3.6]
	1/23	1/24	Local Search	[Ch 4.1-4.2]
	1/25		Project 1 Out	
4	1/28	1/29	Adversarial Search	[Ch 5.1-5.3]
	1/30	1/31	Knowledge Based Agents	[Ch 7.1-7.3]
5	2/4	2/5	Propositional Logic Inference	[Ch 7.4-7.5]
	2/6	2/7	First-Order Logic	[Ch 8.1-8.4]
	2/8		Project 1 Due	
	2/8		Homework 1 Out	
6	2/11	2/12	Rule-Based Systems	[Ch 9.3-9.4]
	2/13	2/14	Search-Based Planning	[Ch 10.1-10.3]
	2/15		Homework 1 Due	
7	2/18	2/19	SAT-Based Planning	[Ch 10.4]
	2/20	2/21	Knowledge Representation	[Ch 12.1-12.5]
8	2/25	2/26	Midterm Review	
	2/27	2/28	**Midterm Exam**	

Outline

- What is AI?
- Problem-solving agent
- Uninformed search
- Informed search (A*)
- Local search
- **Adversarial search**
  - The minimax algorithm
  - Alpha-beta pruning

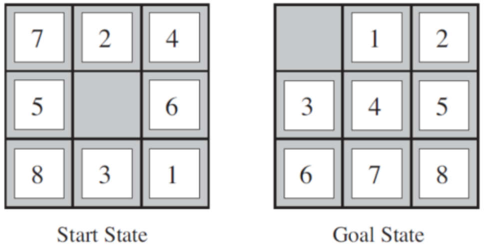
Recap: *When to use “local search”?*

- Solution to some search problem is a “**sequence of actions**” leading to a goal state
  - Example: 8-puzzle

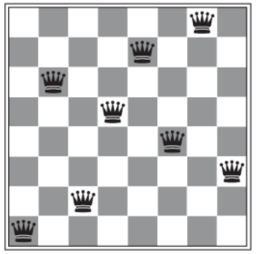


Recap: *When to use "local search"?*

- Solution to some search problem is a "**sequence of actions**" leading to a goal state
  - Example: 8-puzzle



- What if you just want "**goal state**", not "path to goal state"?
  - Example: 8-queens



  - In such cases, you may use "**Local Search**"

Recap: *the idea of local search*

- Operate using a **single, current node** (rather than paths) and move only to **neighbors** of that node

- Advantages
  - Use very little memory
  - Often find reasonable solutions in large and infinite spaces

Recap: *Hill-climbing search*

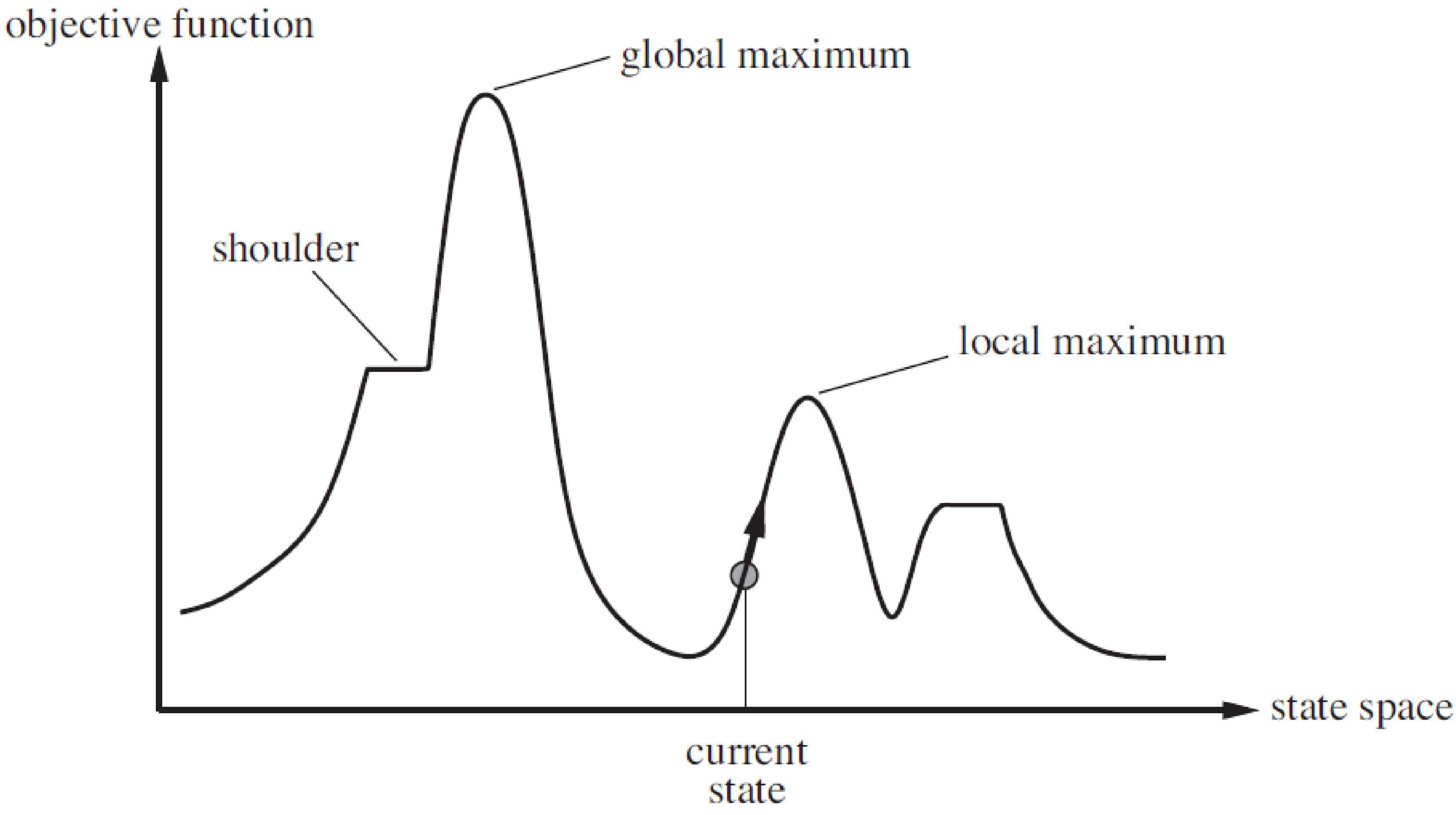
- Continually moves in the direction of increasing value (***steepest-ascent*** version)

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

$current \leftarrow$ MAKE-NODE($problem$.INITIAL-STATE)
loop do
  $neighbor \leftarrow$ a highest-valued successor of $current$
  if neighbor.VALUE $\leq$ current.VALUE **then return** $current$.STATE
  $current \leftarrow neighbor$

Recap: *the state space landscape*

- *Local maximum vs. global maximum*



Recap: *Simulated annealing*

- Combines "**hill climbing**" with "**random walk**" in a way that yields both efficiency and completeness

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(*problem*.INITIAL-STATE)
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow next.\text{VALUE} - current.\text{VALUE}$
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

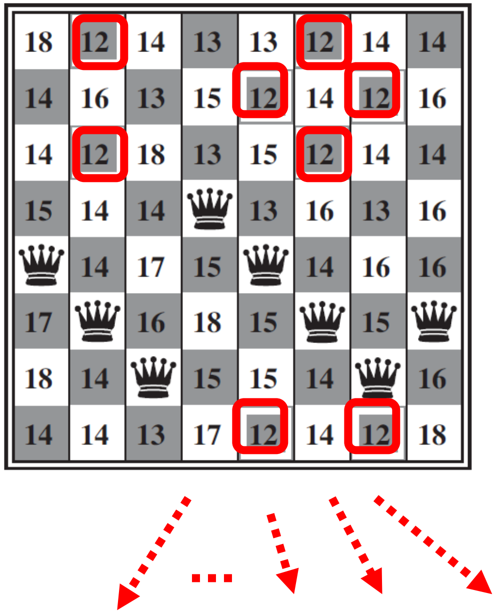
T decreases over time

"hill climbing"

"random walk"

Recap: *Local beam search*

- Keep track of (k) states rather than one
  - Begin with (k) randomly generated states
  - At each step, successors of (k) states are generated
  - Select (k) best successors, and
  - Repeat



Recap: *Genetic algorithm*

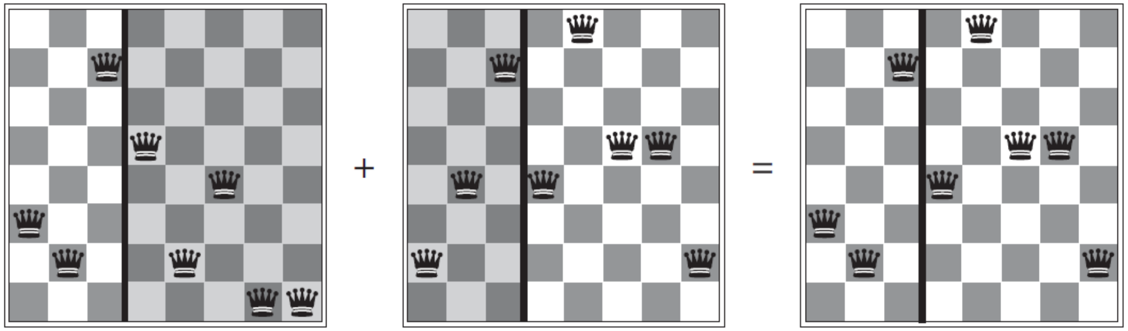
Cross-over + Mutation + Selection

=

Genetic Algorithm

Recap: *Genetic algorithm -- Cross-over*

- Create a state by combining components of two states

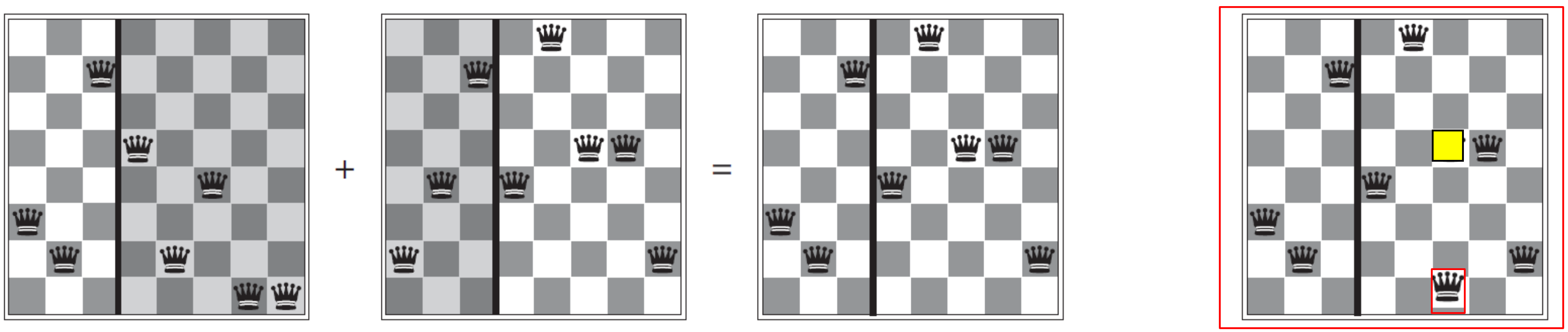


32752411 + 247**48552** = **32748552**

Problem: Lack of diversity (new components)?

Recap: *Genetic algorithm -- Mutation*

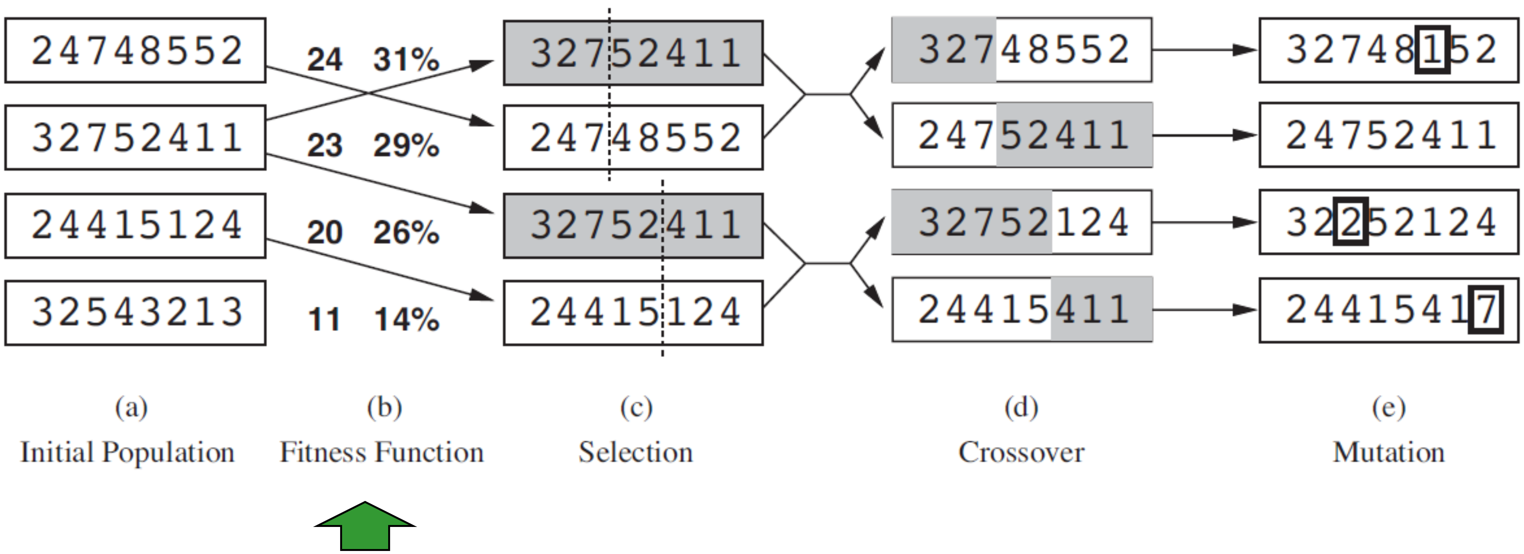
- Randomly change the position of a queen



32752411 + 24748552 = 32748552 ➔ 32748152

Recap: *Genetic algorithm -- Selection*

- Use most promising states for "cross-over" and "mutation" -- they form a population



Outline

- What is AI?
- Problem-solving agent
- Uninformed search
- Informed search (A*)
- **Local search**
  - Hill-climbing search
  - Simulated annealing
  - Local beam search
  - Genetic algorithm
  - **Local search in continuous space**

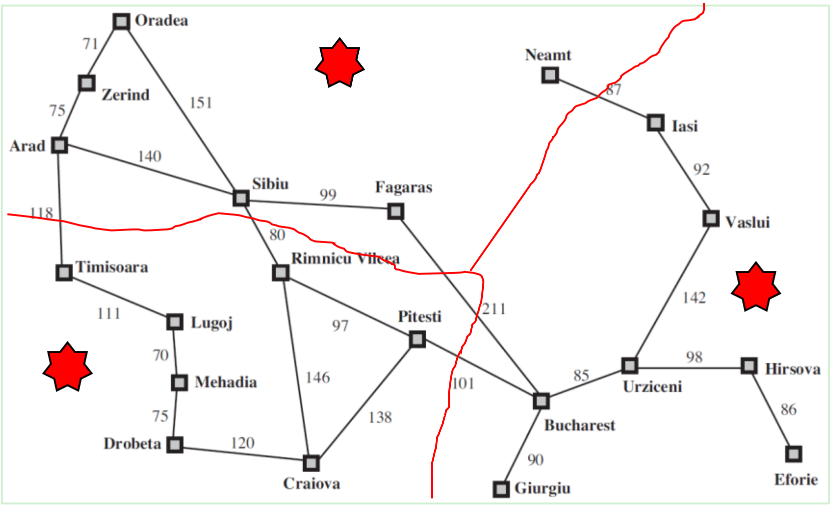
Discrete space vs. continuous space

- Most real-world environments are continuous, but none of the local search algorithms described so far can handle continuous space
  - Continuous **state** space
  - Continuous **action** space

- What could be the problem?
  - "Infinite" branching factors

Airport placement problem

- Placing 3 new airports in Romania to minimize **the sum of** **squared distances** from each city to its nearest airport
  - Coordinates of the airports
    - (x1,y1)
    - (x2,y2)
    - (x3,y3)
  - C1/C2/C3 = sets of cities closet to airport 1/2/3



  - Sum of squared distances

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

Finding a minimum in continuous space

$$\boxed{f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2}$$

- **Option #1**: Discretizing the continuous space

Finding a minimum in continuous space

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

- **Option #2**: Using the "**gradient**" of the landscape
  - The gradient is a vector that gives the magnitude and direction of the steepest slope

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

  - The minimum can be find by solving the equation

$$\nabla f = 0$$

The concavity of a function is given by its second derivative: A positive second derivative means the function is concave up (minimum), a negative second derivative means the function is concave down (maximum), and a second derivative of zero is inconclusive.

Finding a minimum *(computationally)*

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

- Given the gradient, we can perform the "steepest-ascent" hill-climbing search

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$$

$$\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_i - x_c)$$

Step size

Finding a minimum *(computationally)*

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

- Given the gradient, we can perform the "steepest-ascent" hill-climbing search

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x})$$

$$\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_i - x_c)$$

$\mathbf{H}_f(\mathbf{x})$ is the Hessian matrix of second derivatives

$$\partial^2 f / \partial x_i \partial x_j$$

Outline

- What is AI?
- Problem-solving agent
- Uninformed search
- Informed search (A*)
- Local search
- **Adversarial search**
  - – The minimax algorithm
  - – Alpha-beta pruning

Game vs. search problem

- So far, problems have been deterministic with a known transition model, which represents the environment
- **Adversarial:** there is another agent working against you
  - The opponent is not predictable
  - Can’t be represented by a transition model
- Game trees instead of search trees
  - Solution is no longer a “sequence of actions” but a **contingency plan**

Two-player, turn-based game

- **Players**: MAX and MIN

- **Initial State**:
- **Actions**:
- **Transition Model**:
- **Terminal Test**:
- **Utility Function:**

Two-player, turn-based game

- **Players**: MAX and MIN

- **Initial State**: board position and turn
- **Actions**: set of legal moves in a state
- **Transition Model**: s ← RESULT(s, a)
- **Terminal Test**: condition for when game is over
- **Utility Function:** numeric value that describes outcome

- ***Win:*** ***+1***
- ***Loss:*** ***0***
- ***Draw:*** ***1/2***

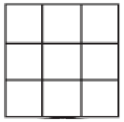
"**Zero-Sum**" Game

win: +1/2
loss: -1/2
draw: 0

Game Tree (tic-tac-toe)

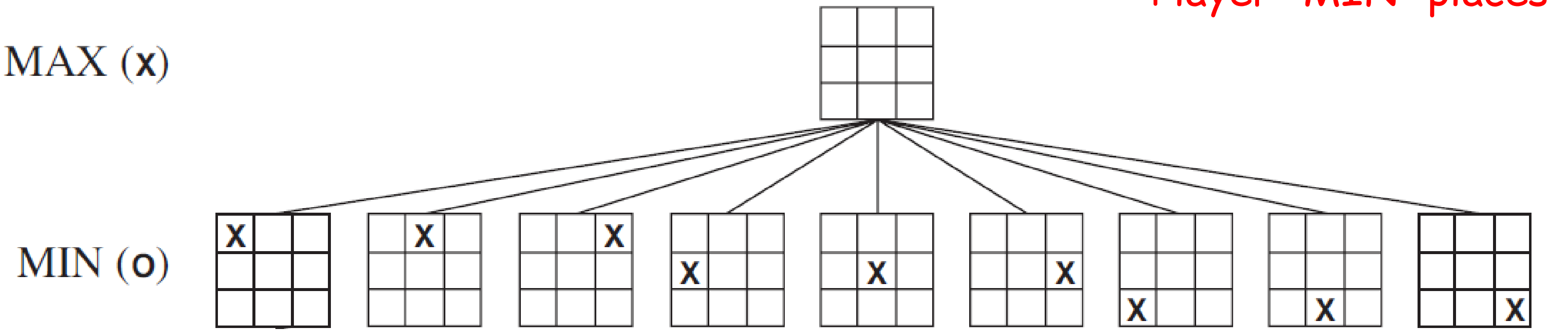
Player "MAX" places an "x"

MAX (**x**)



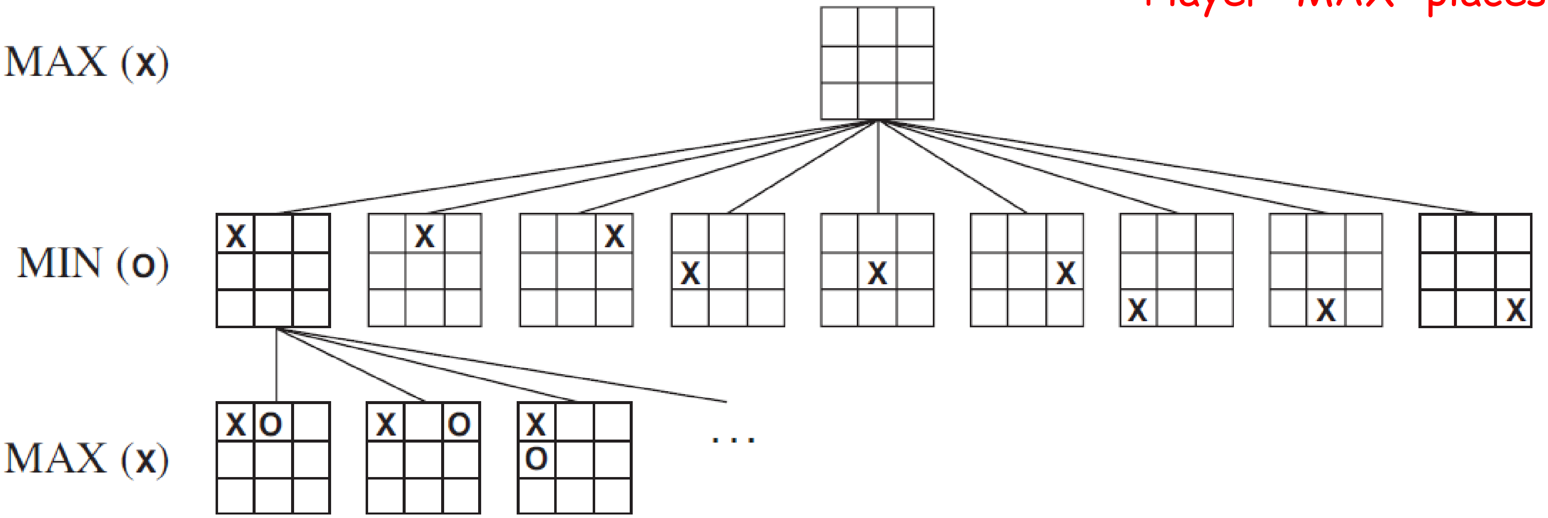
Game Tree (tic-tac-toe)

Player "MIN" places an "o"



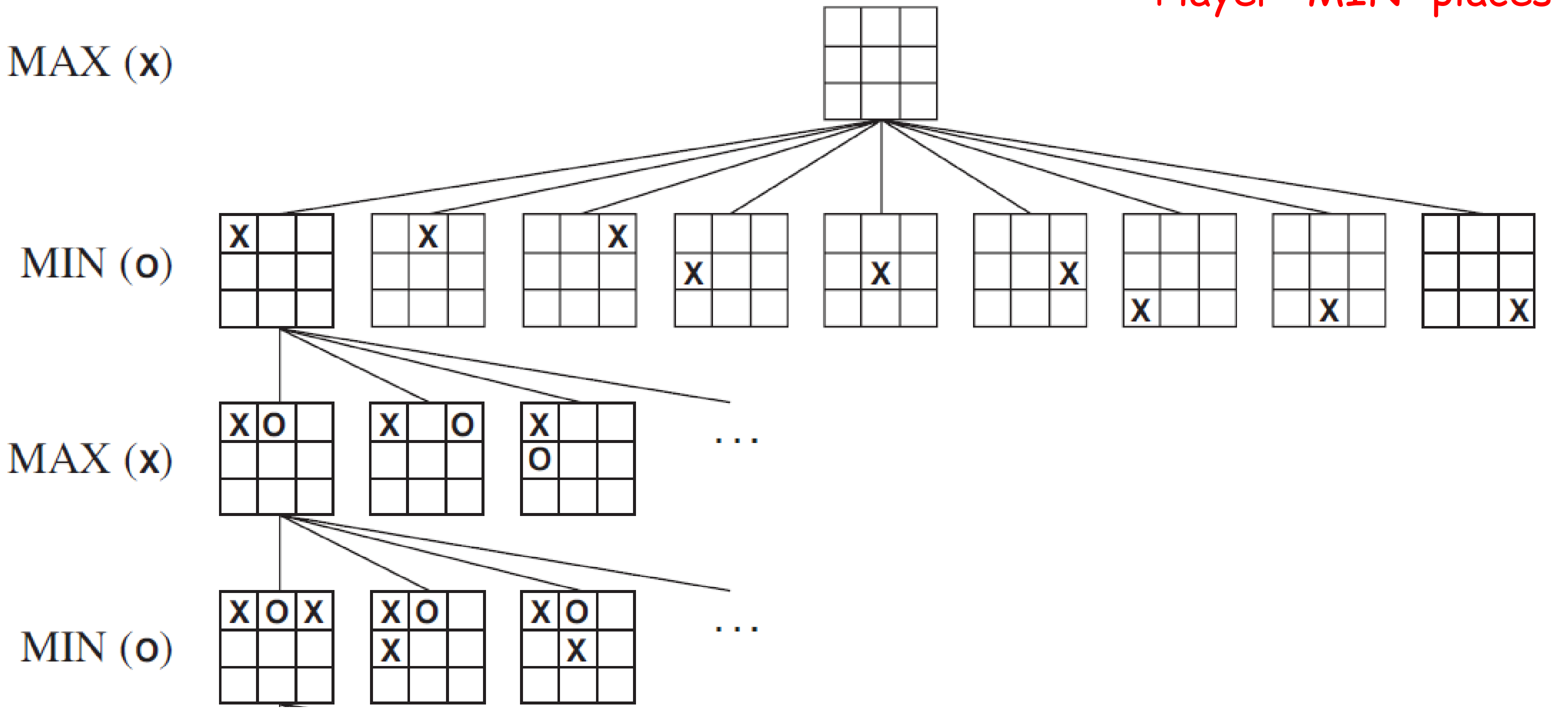
Game Tree (tic-tac-toe)

Player "MAX" places an "x"

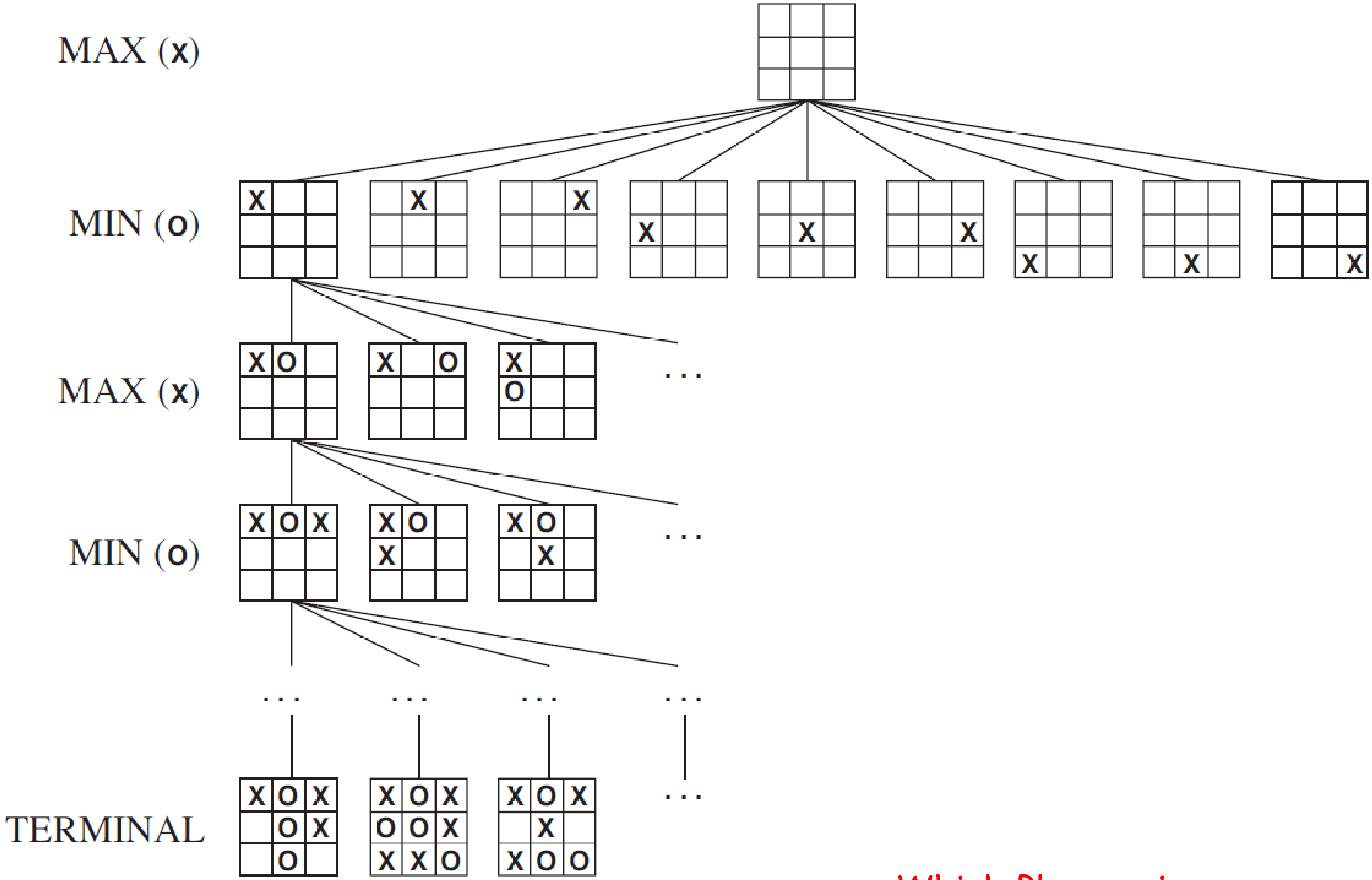


Game Tree (tic-tac-toe)

Player "MIN" places an "o"

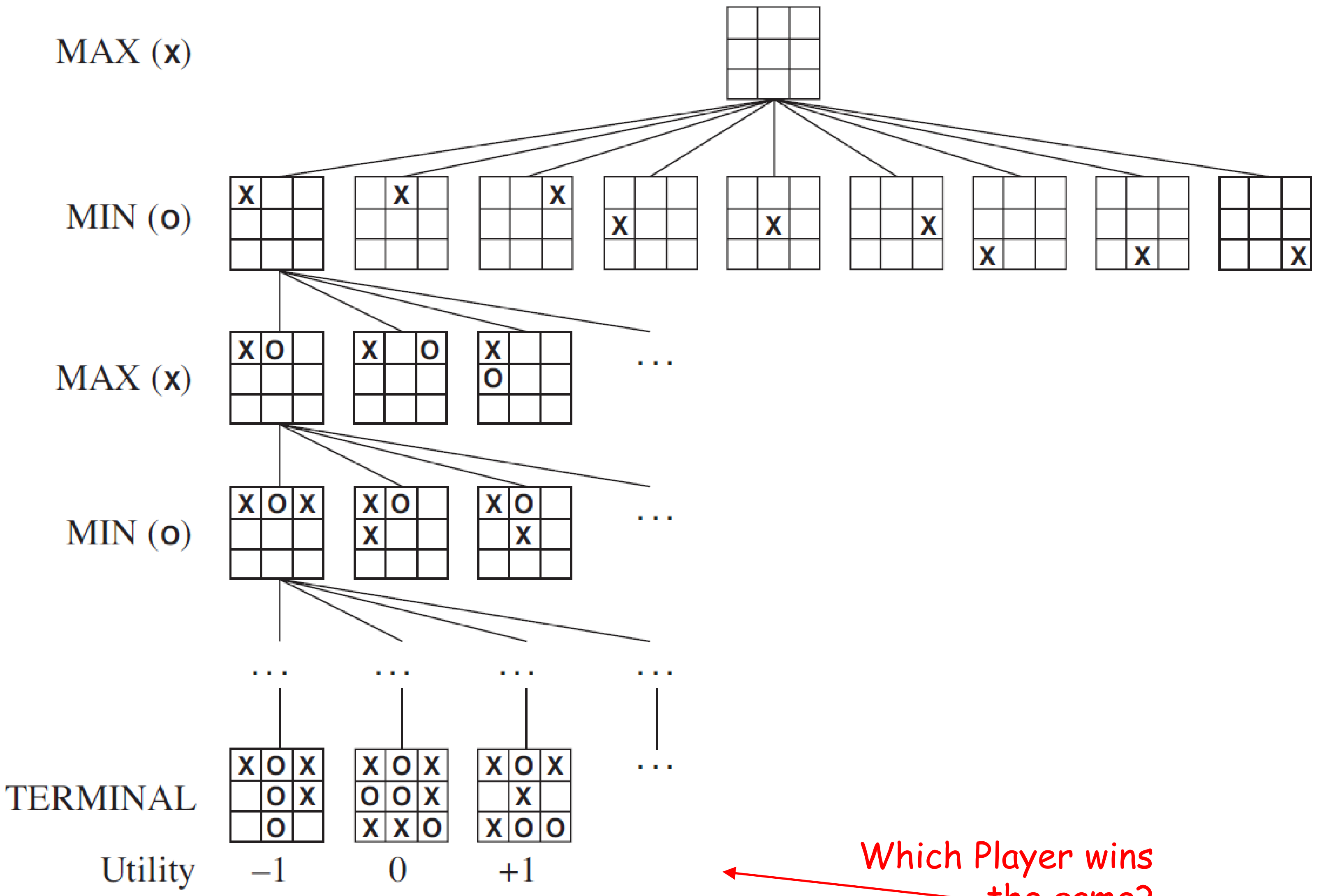


Game Tree (tic-tac-toe)

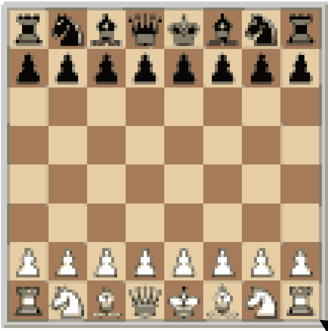


Which Player wins
the game?

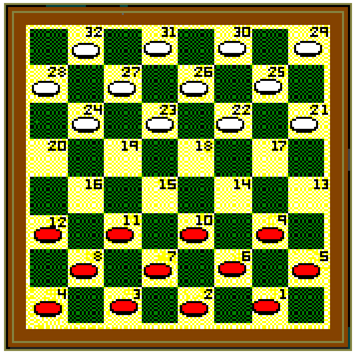
Game Tree (tic-tac-toe)



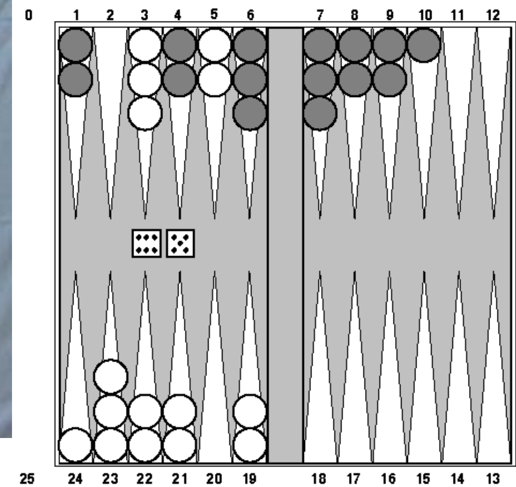
Which Player wins the game?



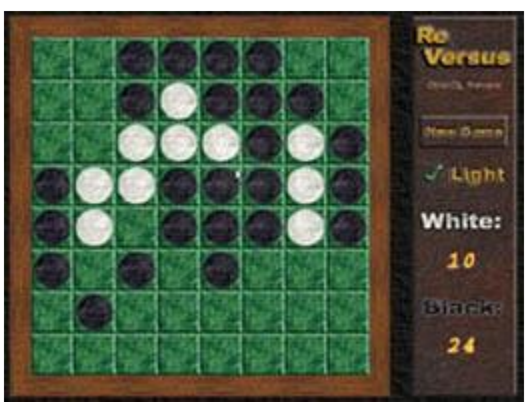
The board set for play



Red to play



	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information		bridge, poker, scrabble



Minimax

- Perfect play for deterministic environment with perfect information
- **Basic idea**: Choose move with ***highest minimax value***
  - = best achievable **payoff** against perfect opponent

Minimax

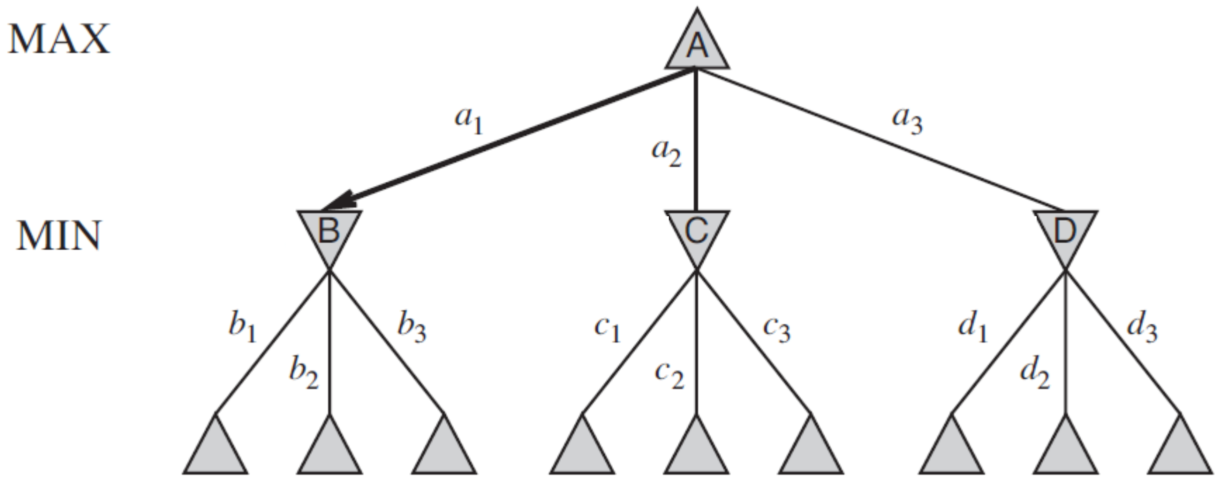
- Perfect play for deterministic environment with perfect information
- **Basic idea**: Choose move with ***highest minimax value***
  - = best achievable **payoff** against perfect opponent

Algorithm:

1. Generate the entire game tree
2. Determine utility of each terminal state
3. Back propagate utility values in tree to compute minimax values
4. At the root note, choose move with highest minimax value

Solution of a game

- It specifies
  - MAX’s move in the initial state, then
  - MAX’s moves in states resulting from every possible response by MIN, then
  - MAX’s moves in the states resulting from every possible response by MIN to those moves of MAX,
  - …



Minimax value of each node

- Utility (for MAX) of being in the state, assuming that both players play optimally from there to the end of the game

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

MAX wants to maximize the utility

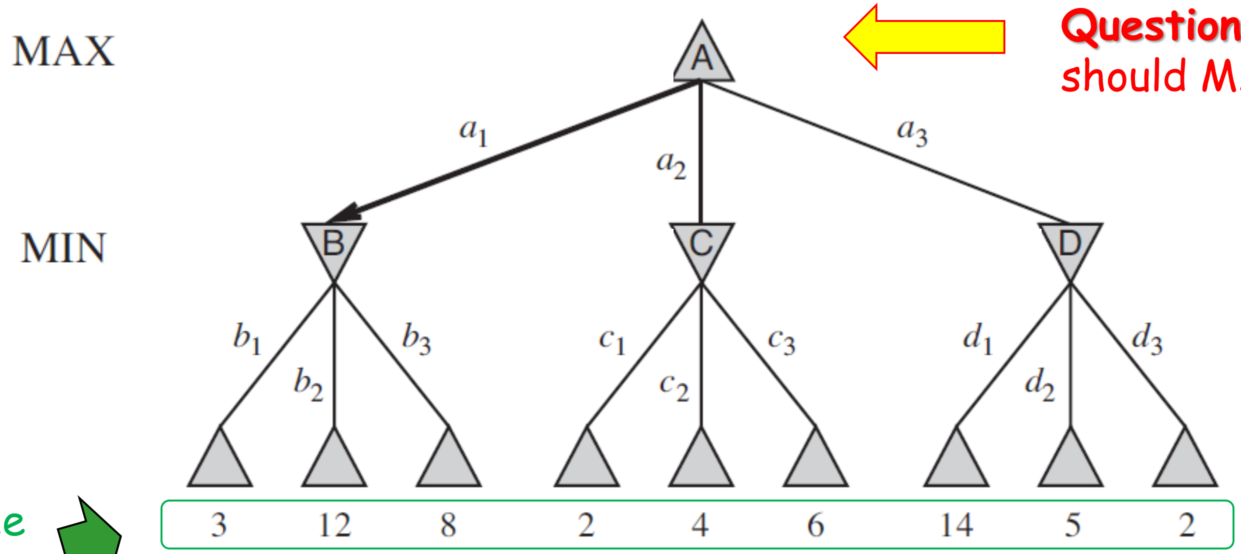
MIN wants to minimize the utility

Minimax value of each node

- Utility (for MAX) of being in the state, assuming that both players play optimally from there to the end of the game

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

Question: Which action should MAX take?

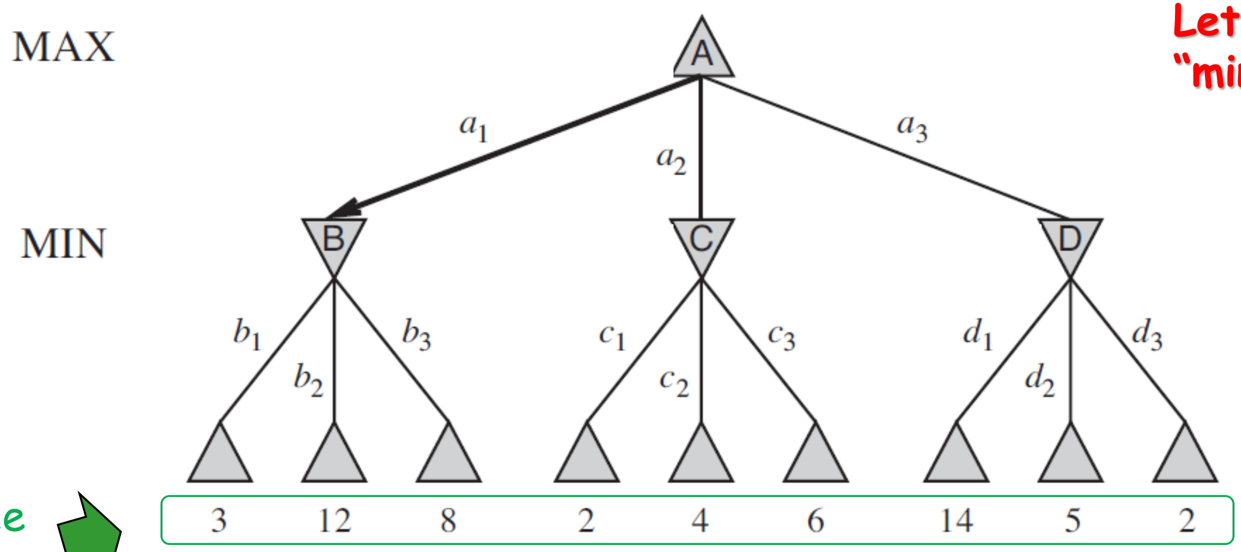


MAX's chance of winning

Minimax value of each node

- Utility (for MAX) of being in the state, assuming that both players play optimally from there to the end of the game

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



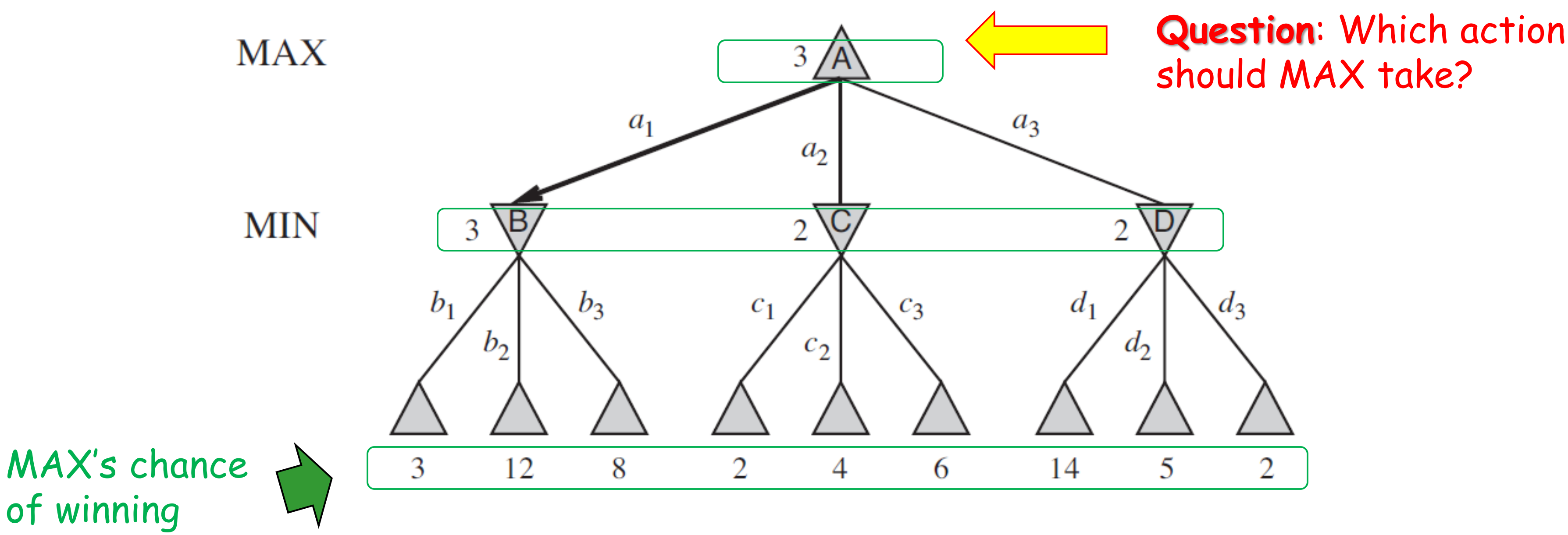
Let's compute the "minimax" value

MAX's chance of winning

Minimax value of each node

- Utility (for MAX) of being in the state, assuming that both players play optimally from there to the end of the game

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



Minimax algorithm

Choose the action of MAX that "**maximize**" this value

function MINIMAX-DECISION(*state*) **returns** *an action*
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ 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*)
  $v \leftarrow \infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(*s*, *a*)))
  return v

Both values are utility for Player MAX

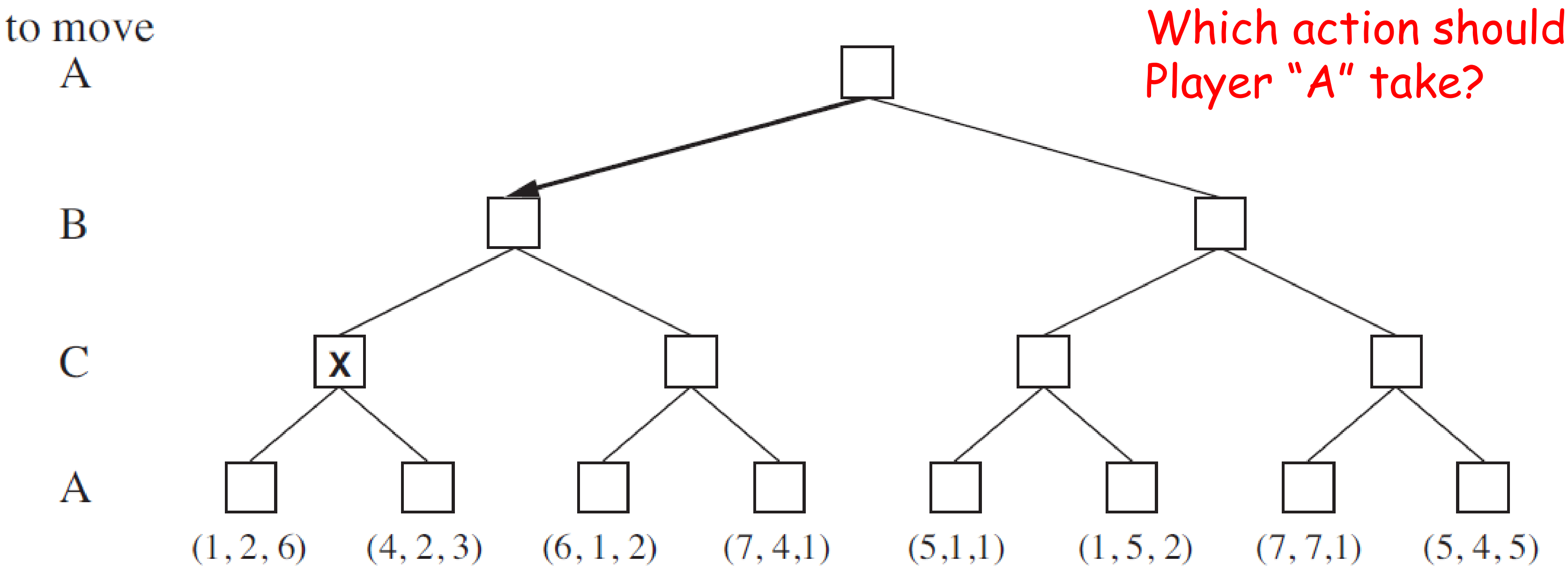
We don't need the utility for Player MIN because this is a "zero-sum" game

3-player game?

- Instead of a single (utility) value for Player MAX, we need a vector of three values
  - (v_A, v_B, v_C)
  - Each value gives the utility of a state from one player's viewpoint

3-player game?

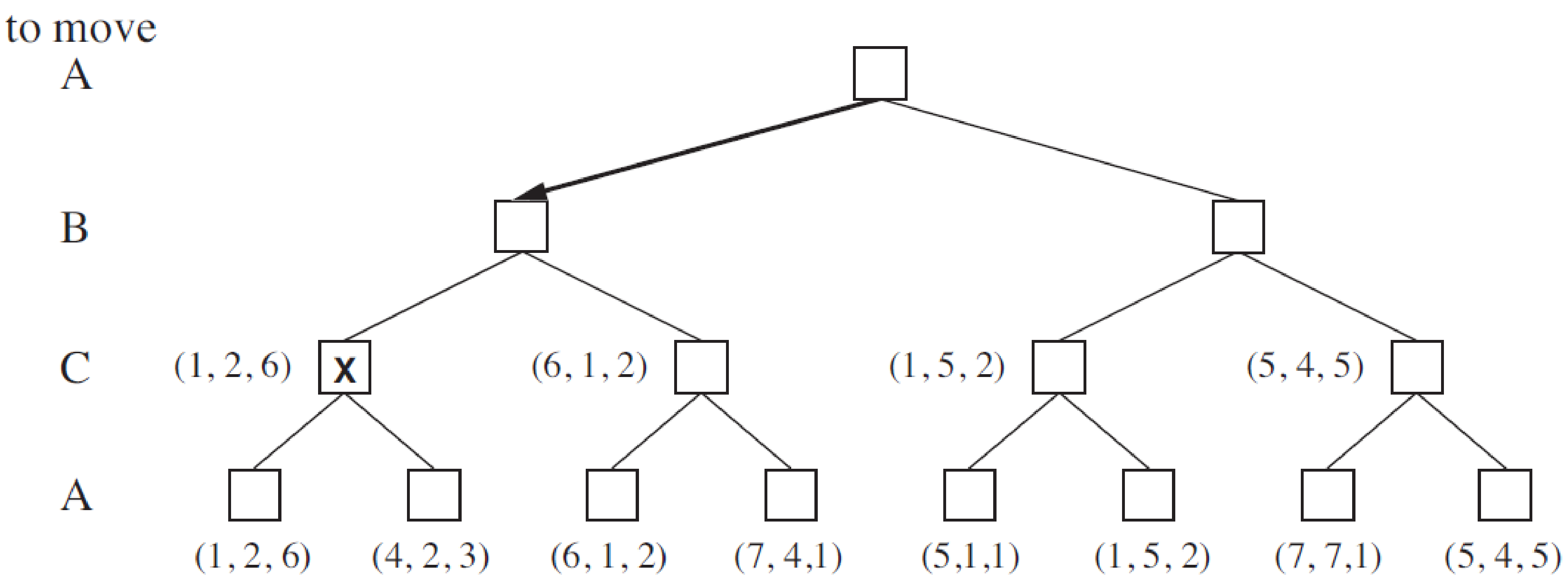
- Instead of a single (utility) value for Player MAX, we need a vector of three values
  - (v_A, v_B, v_C)
  - Each value gives the utility of a state from one player's viewpoint



Which action should Player "A" take?

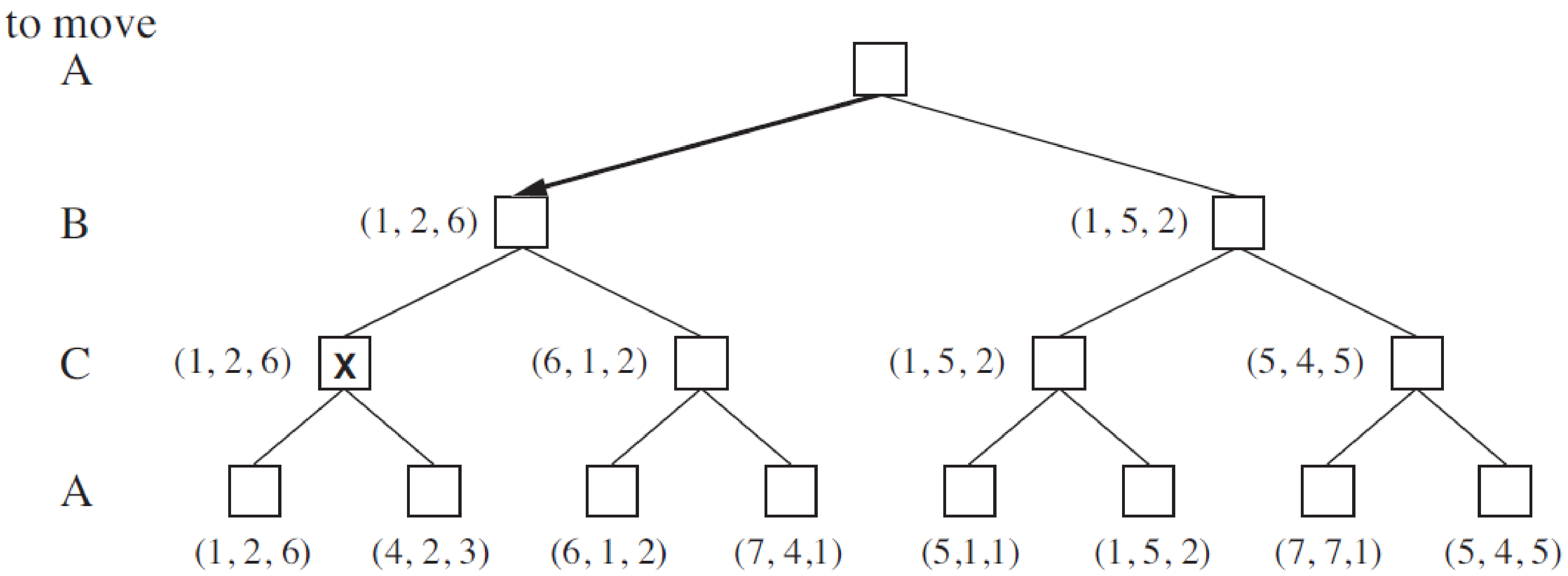
3-player game?

- Instead of a single (utility) value for Player MAX, we need a vector of three values
  - (v_A, v_B, v_C)
  - Each value gives the utility of a state from one player's viewpoint



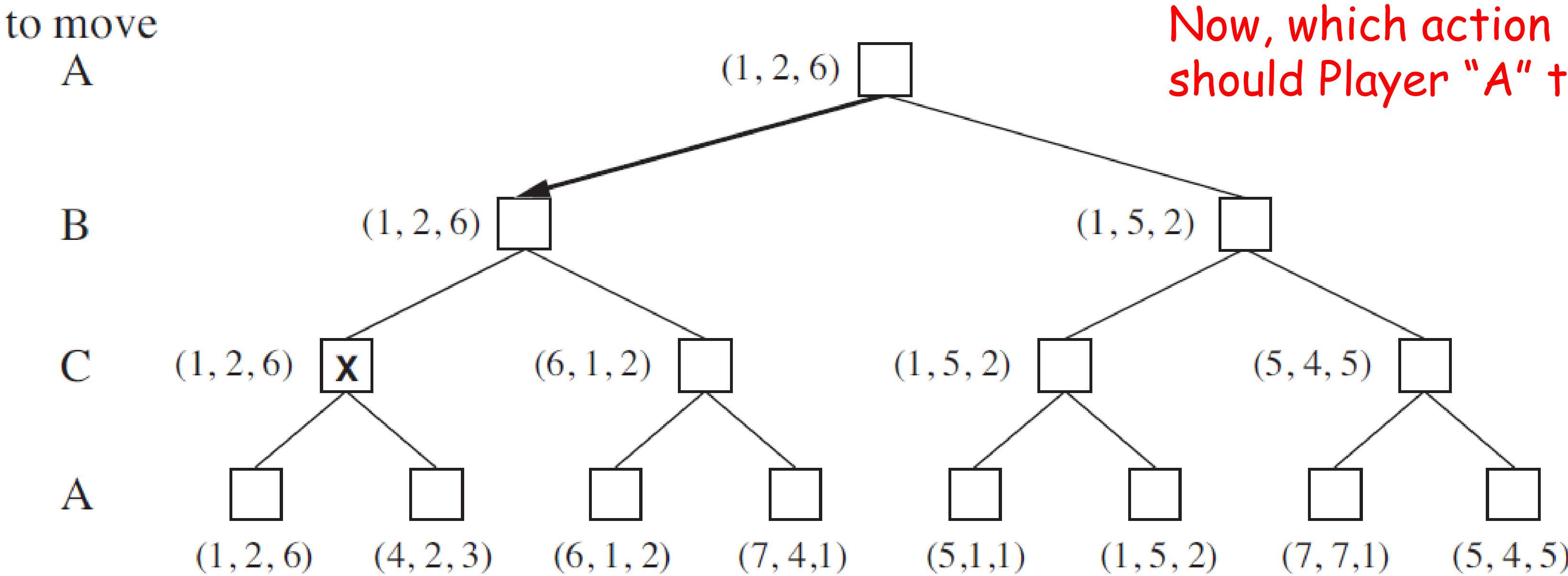
3-player game?

- Instead of a single (utility) value for Player MAX, we need a vector of three values
  - (v_A, v_B, v_C)
  - Each value gives the utility of a state from one player's viewpoint



3-player game?

- Instead of a single (utility) value for Player MAX, we need a vector of three values
  - (v_A, v_B, v_C)
  - Each value gives the utility of a state from one player's viewpoint



Now, which action should Player "A" take?

Properties of Minimax

- performs a complete depth-first search of the game tree

- **Complete?** Yes (if game three is finite)
- **Optimal?** Yes (against optimal opponent)

- **Time complexity**: $O(b^m)$ — b = legal move, m = maximum depth
- **Space complexity**: $O(bm)$

Complexity of Minimax

- Performs a complete depth-first search of the game tree

- **Time complexity**: ***$O(b^m)$***   ***b*** = legal move
  m = maximum depth

The worst-case exponential complexity is unavoidable.
But, can we do "**somewhat**" better **in practice**?

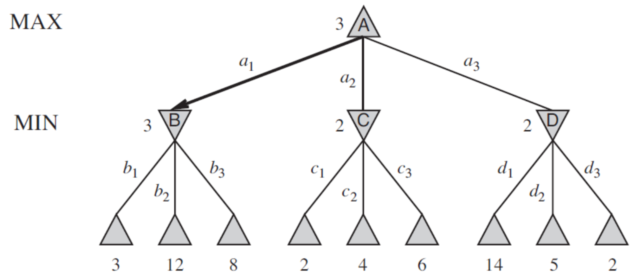
Outline

- What is AI?
- Problem-solving agent
- Uninformed search
- Informed search (A*)
- Local search
- **Adversarial search**
  - The minimax algorithm
  - Alpha-beta pruning

Alpha-beta pruning

- Don’t explore branches of the game tree that cannot lead to a better outcome (than those that have been explored)

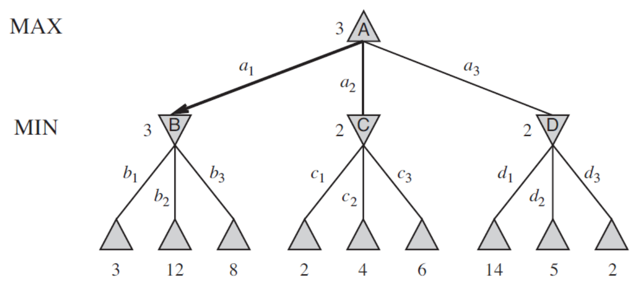
$$\boxed{\textsc{Minimax}(root) \ = \ \max(\min(3,12,8), \min(2,x,y), \min(14,5,2))}$$



Alpha-beta pruning

- Don’t explore branches of the game tree that cannot lead to a better outcome (than those that have been explored)

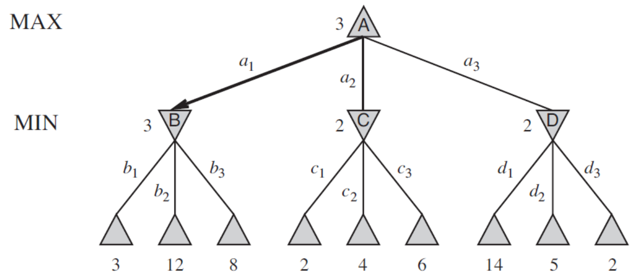
$$\begin{aligned}\textsc{Minimax}(root) &= \max(\min(3,12,8),\min(2,x,y),\min(14,5,2)) \\ &= \max(3,\min(2,x,y),2)\end{aligned}$$



Alpha-beta pruning

- Don't explore branches of the game tree that cannot lead to a better outcome (than those that have been explored)

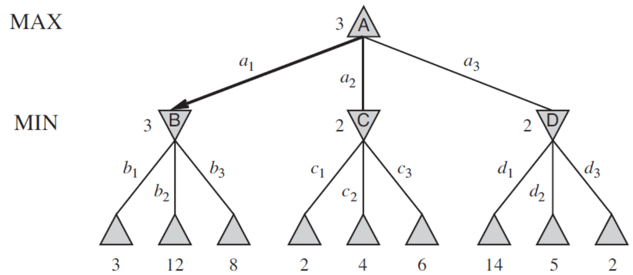
$$\begin{aligned}\textsc{Minimax}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\ &= \max(3, \min(2,x,y), 2) \\ &= \max(3, z, 2) \qquad \text{where } z = \min(2,x,y) \leq 2\end{aligned}$$



Alpha-beta pruning

- Don't explore branches of the game tree that cannot lead to a better outcome (than those that have been explored)

$$\begin{aligned}
\textsc{Minimax}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3, z, 2) \qquad \text{where } z = \min(2,x,y) \leq 2 \\
&= 3.
\end{aligned}$$



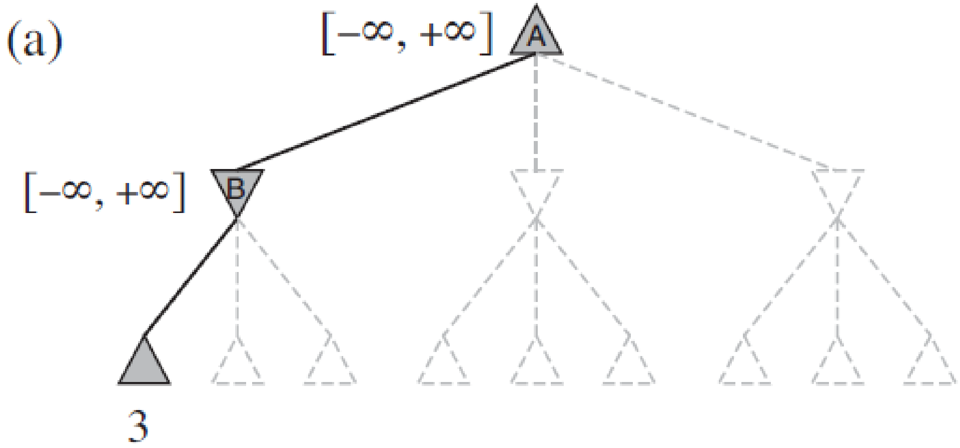
The value of the root (and hence the minimax decision) are independent of the values of the "pruned" leaves x and y

What are “alpha” and “beta”?

- **Alpha** = value of the best (highest-value) choice for ***MAX*** that has been found so far
  - Initialized to -∞, and then keeps increasing
- **Beta** = value of the best (lowest-value) choice of ***MIN*** that has been found so far
  - Initialized to +∞, and then keeps decreasing

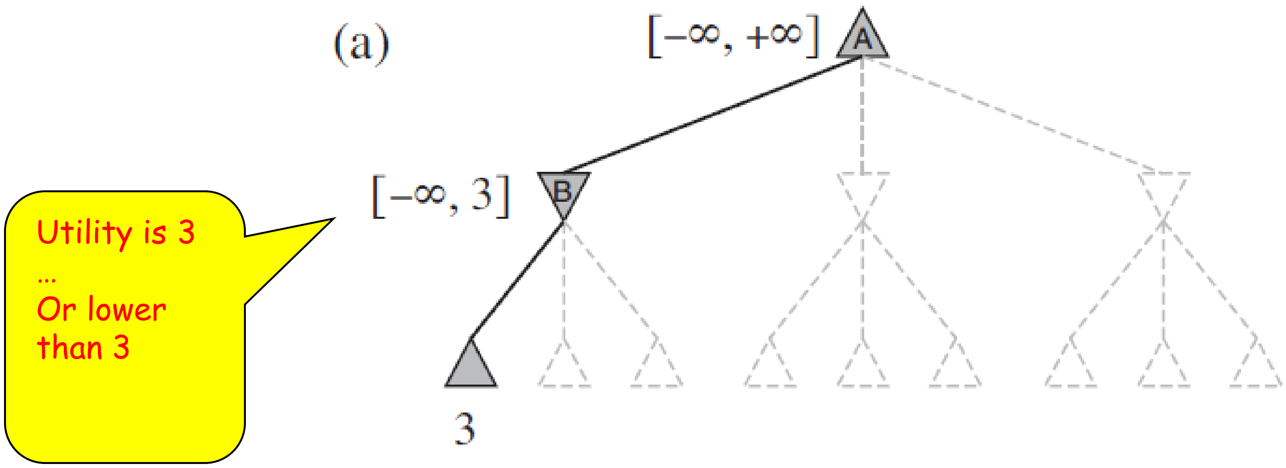
Alpha-beta pruning (example run)

- **Alpha** = value of the best (highest-value) choice for ***MAX*** that has been found so far
  - Initialized to -∞, and then keeps increasing
- **Beta** = value of the best (lowest-value) choice of ***MIN*** that has been found so far
  - Initialized to +∞, and then keeps decreasing



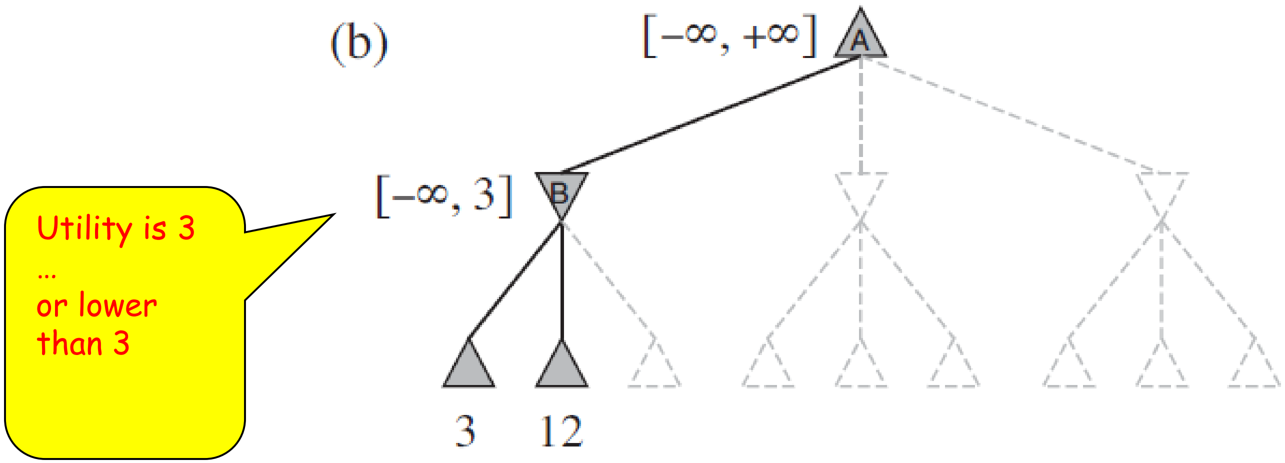
Alpha-beta pruning (example run)

- **Alpha** = value of the best (highest-value) choice for ***MAX*** that has been found so far
  - Initialized to -∞, and then keeps increasing
- **Beta** = value of the best (lowest-value) choice of ***MIN*** that has been found so far
  - Initialized to +∞, and then keeps decreasing



Alpha-beta pruning (example run)

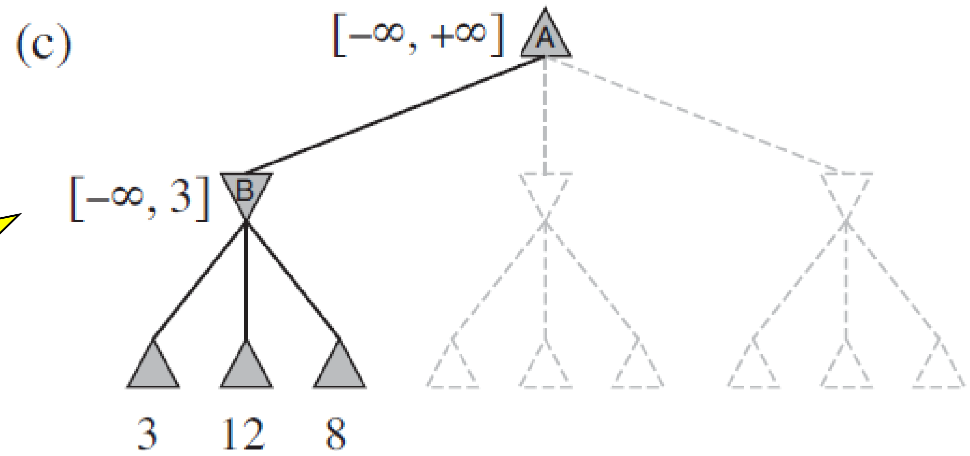
- **Alpha** = value of the best (highest-value) choice for ***MAX*** that has been found so far
  - Initialized to -∞, and then keeps increasing
- **Beta** = value of the best (lowest-value) choice of ***MIN*** that has been found so far
  - Initialized to +∞, and then keeps decreasing



Alpha-beta pruning (example run)

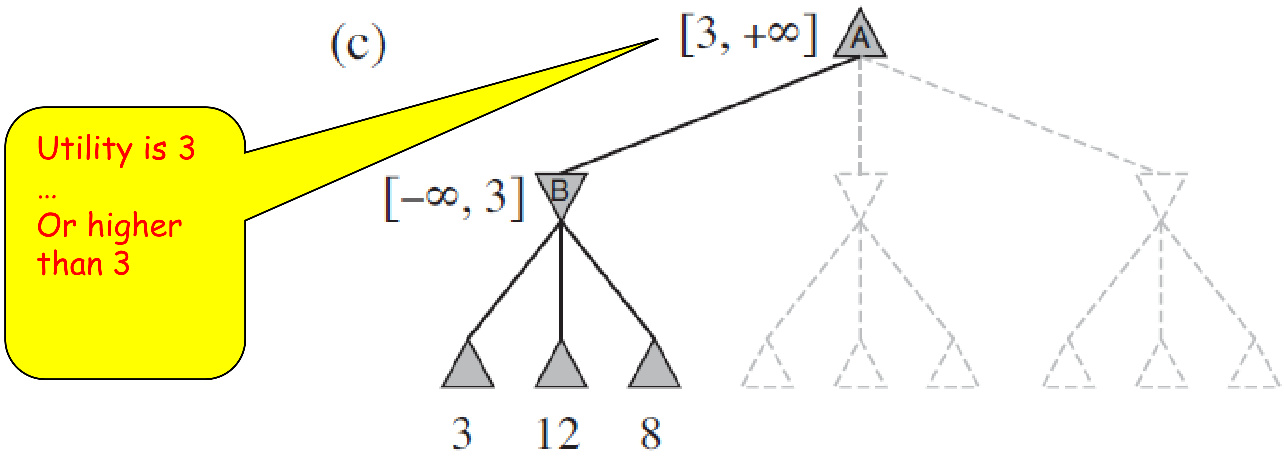
- **Alpha** = value of the best (highest-value) choice for ***MAX*** that has been found so far
  - Initialized to -∞, and then keeps increasing
- **Beta** = value of the best (lowest-value) choice of ***MIN*** that has been found so far
  - Initialized to +∞, and then keeps decreasing

Utility is 3
…
And can't
be lower



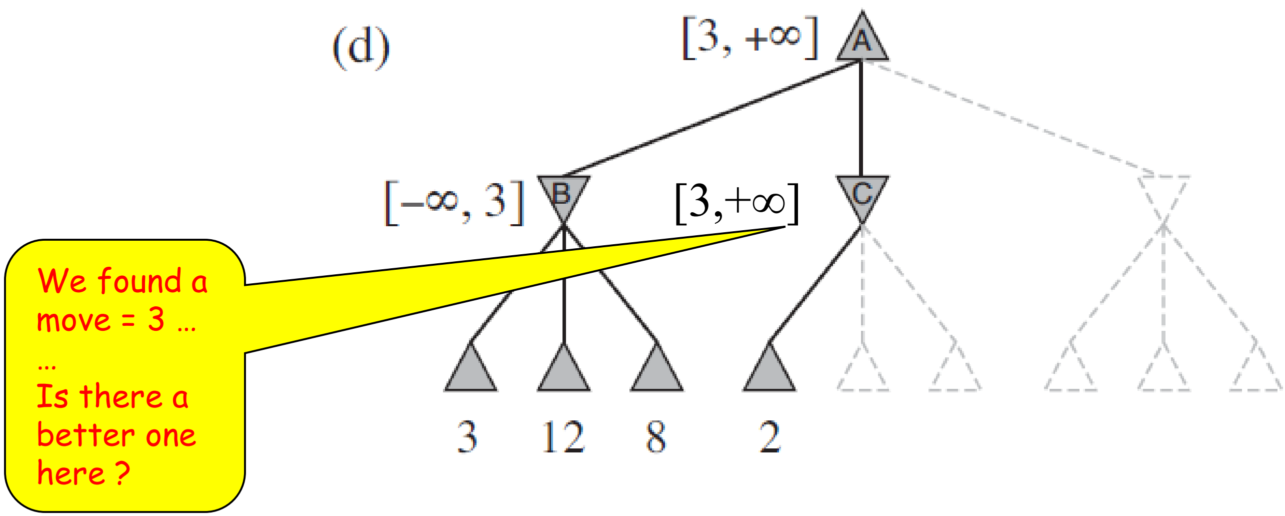
Alpha-beta pruning (example run)

- **Alpha** = value of the best (highest-value) choice for ***MAX*** that has been found so far
  - Initialized to -∞, and then keeps increasing
- **Beta** = value of the best (lowest-value) choice of ***MIN*** that has been found so far
  - Initialized to +∞, and then keeps decreasing



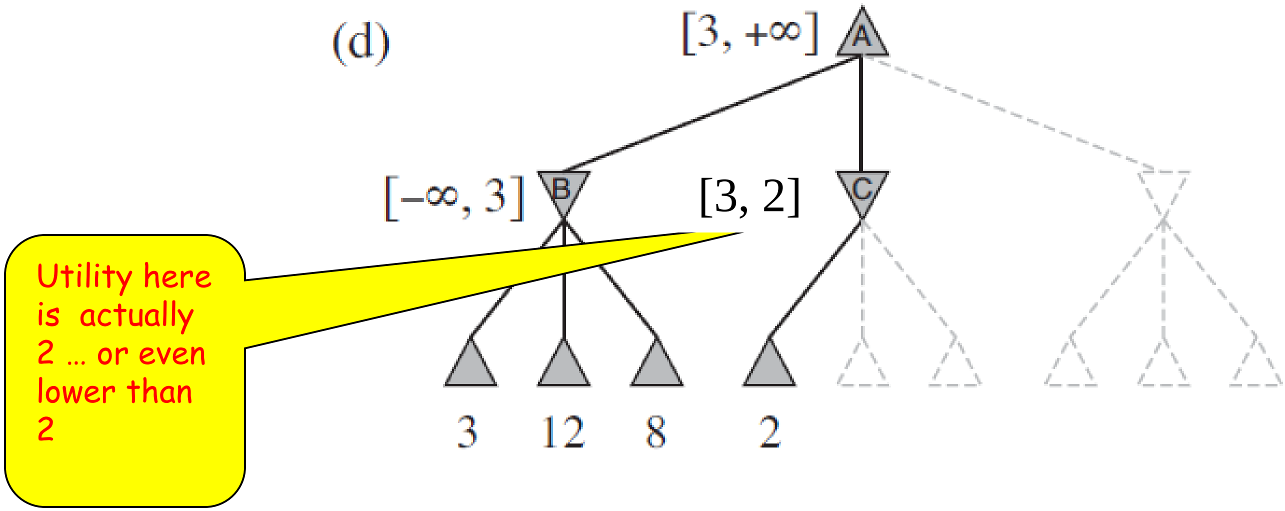
Alpha-beta pruning (example run)

- **Alpha** = value of the best (highest-value) choice for ***MAX*** that has been found so far
  - Initialized to -∞, and then keeps increasing
- **Beta** = value of the best (lowest-value) choice of ***MIN*** that has been found so far
  - Initialized to +∞, and then keeps decreasing



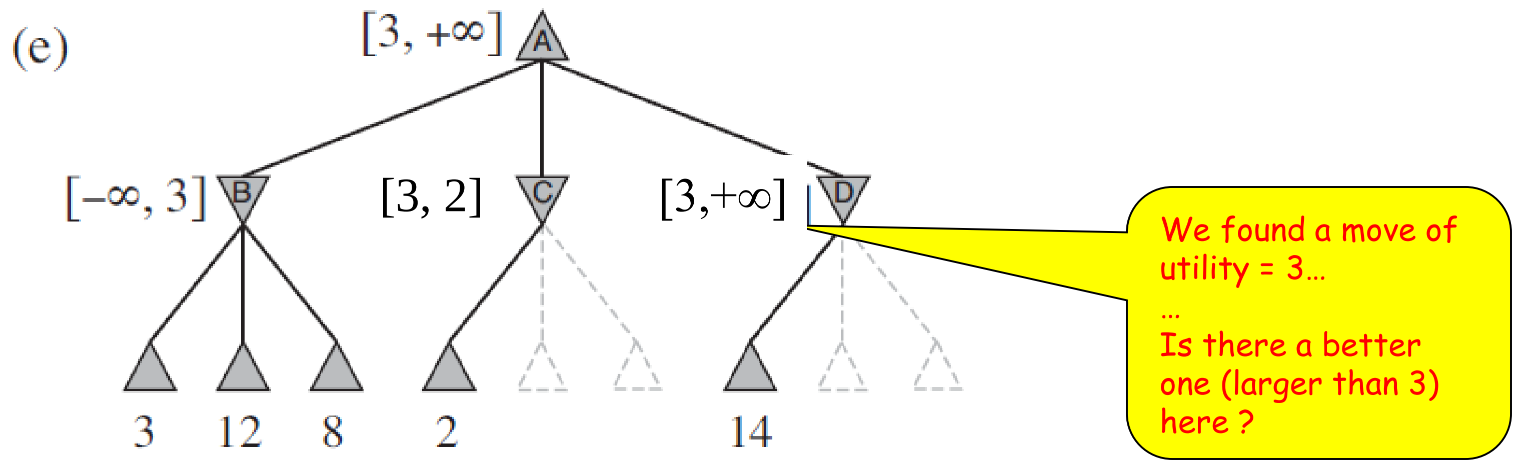
Alpha-beta pruning (example run)

- **Alpha** = value of the best (highest-value) choice for ***MAX*** that has been found so far
  - Initialized to -∞, and then keeps increasing
- **Beta** = value of the best (lowest-value) choice of ***MIN*** that has been found so far
  - Initialized to +∞, and then keeps decreasing



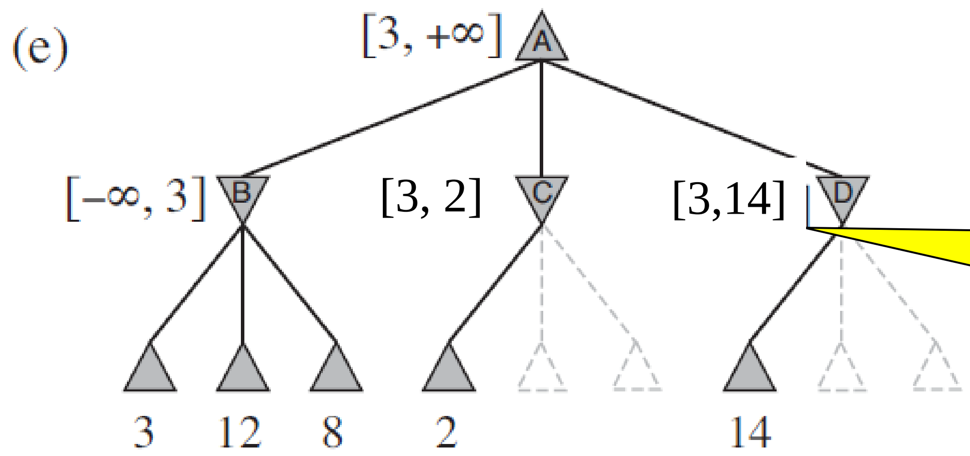
Alpha-beta pruning (example run)

- **Alpha** = value of the best (highest-value) choice for ***MAX*** that has been found so far
  - Initialized to -∞, and then keeps increasing
- **Beta** = value of the best (lowest-value) choice of ***MIN*** that has been found so far
  - Initialized to +∞, and then keeps decreasing



Alpha-beta pruning (example run)

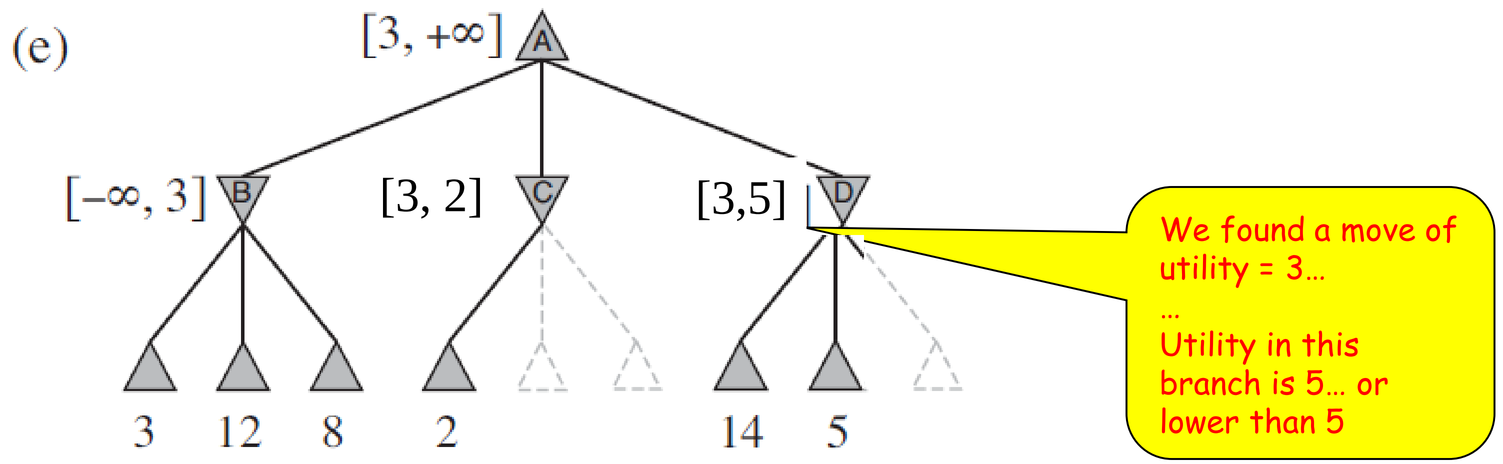
- **Alpha** = value of the best (highest-value) choice for ***MAX*** that has been found so far
  - Initialized to -∞, and then keeps increasing
- **Beta** = value of the best (lowest-value) choice of ***MIN*** that has been found so far
  - Initialized to +∞, and then keeps decreasing



We found a move of utility = 3…
…
Utility in this branch is 14…
… or lower than 14

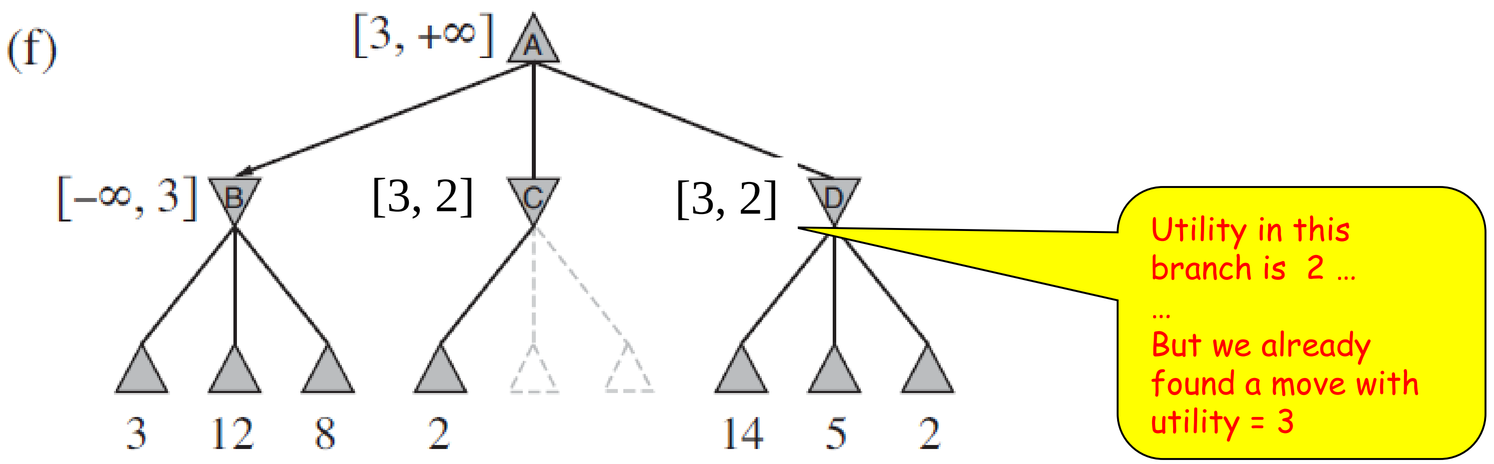
Alpha-beta pruning (example run)

- **Alpha** = value of the best (highest-value) choice for ***MAX*** that has been found so far
  - Initialized to -∞, and then keeps increasing
- **Beta** = value of the best (lowest-value) choice of ***MIN*** that has been found so far
  - Initialized to +∞, and then keeps decreasing



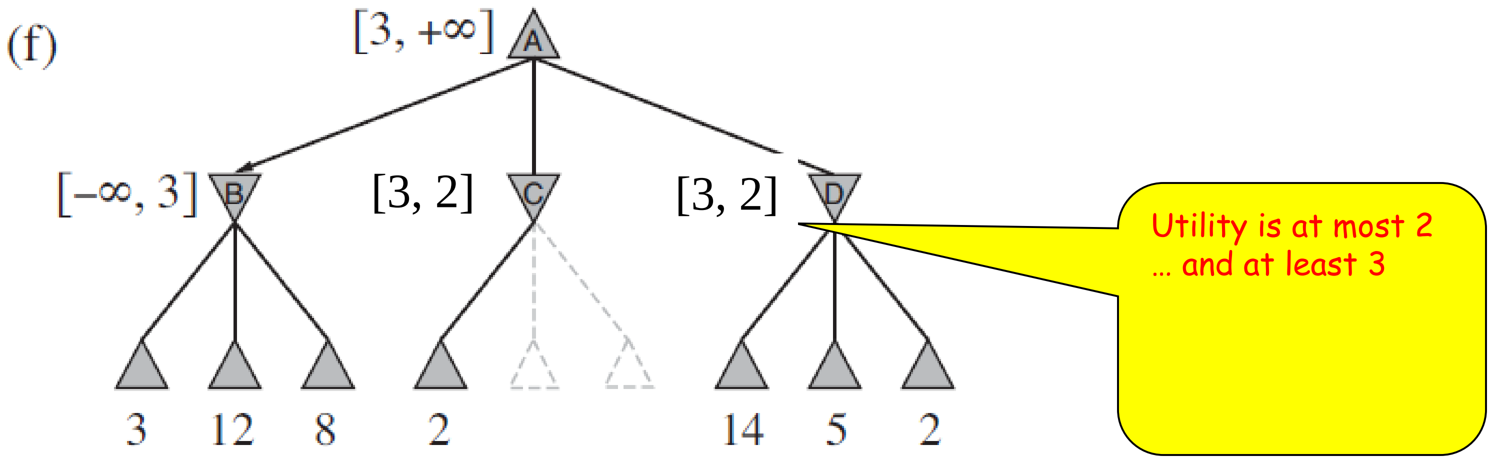
Alpha-beta pruning (example run)

- **Alpha** = value of the best (highest-value) choice for ***MAX*** that has been found so far
  - Initialized to -∞, and then keeps increasing (at least Alpha)
- **Beta** = value of the best (lowest-value) choice of ***MIN*** that has been found so far
  - Initialized to +∞, and then keeps decreasing (at most Beta)



Alpha-beta pruning (example run)

- **Alpha** = value of the best (highest-value) choice for ***MAX*** that has been found so far
  - Initialized to -∞, and then keeps increasing (at least Alpha)
- **Beta** = value of the best (lowest-value) choice of ***MIN*** that has been found so far
  - Initialized to +∞, and then keeps decreasing (at most Beta)



If Player has a better choice ***m***, either at the parent node of ***n***, or at any choice point further up, then ***n*** will never be reached in actual play.

Minimax algorithm

function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

---

function MAX-VALUE($state, \alpha, \beta$) **returns** a $utility$ $value$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
  return v

---

function MIN-VALUE($state, \alpha, \beta$) **returns** a $utility$ $value$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
  return v

Minimax with alpha-beta pruning

function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

---

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

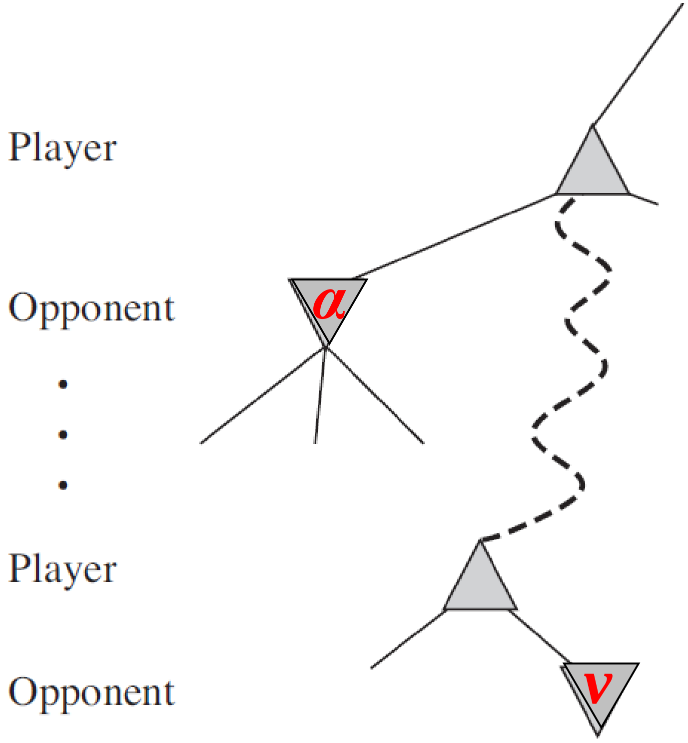
function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a), α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Properties of alpha-beta pruning

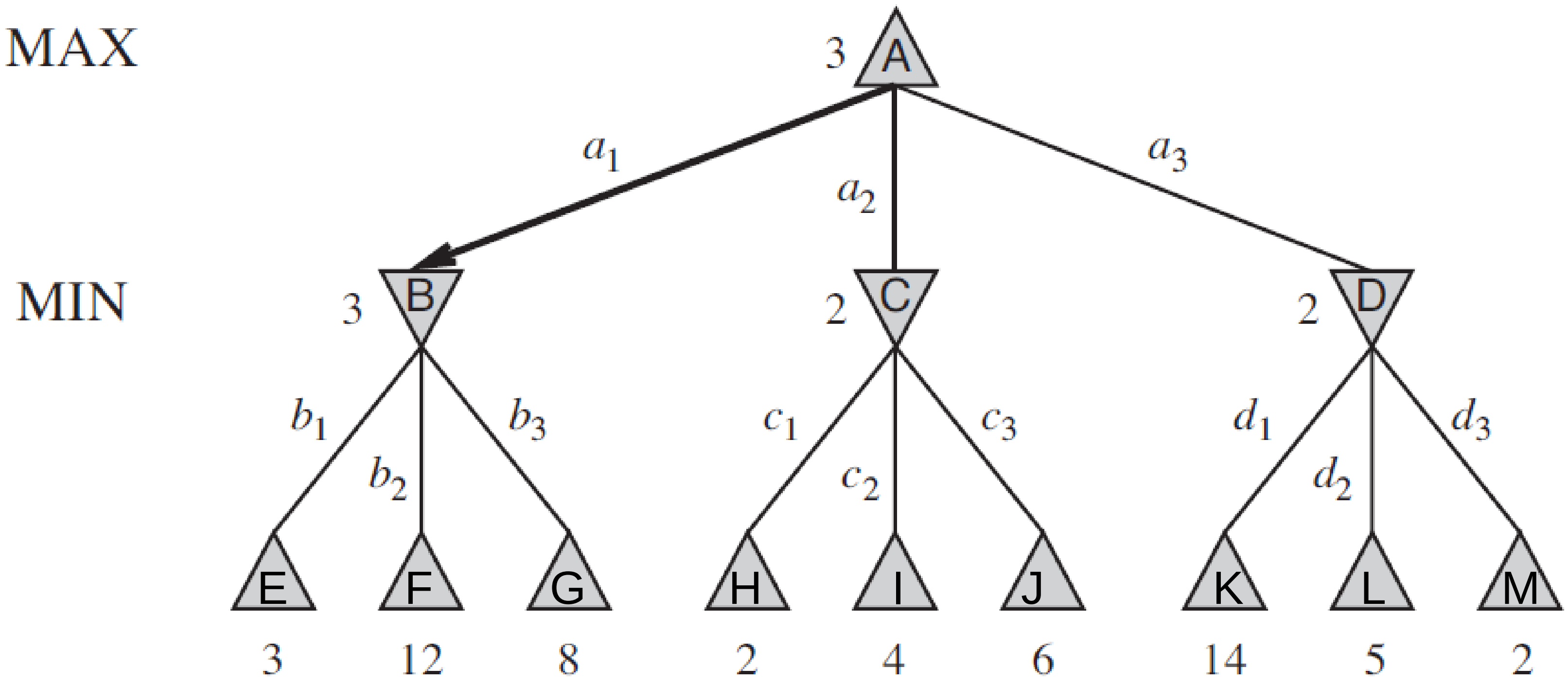
- Pruning does NOT affect the final result of Minimax
- Effectiveness of pruning is highly dependent on **the order** in which states are examined
- **Worst case**: no improvement at all
- **Best case**: $O(b^{m/2})$

Why is it called “alpha-beta”?

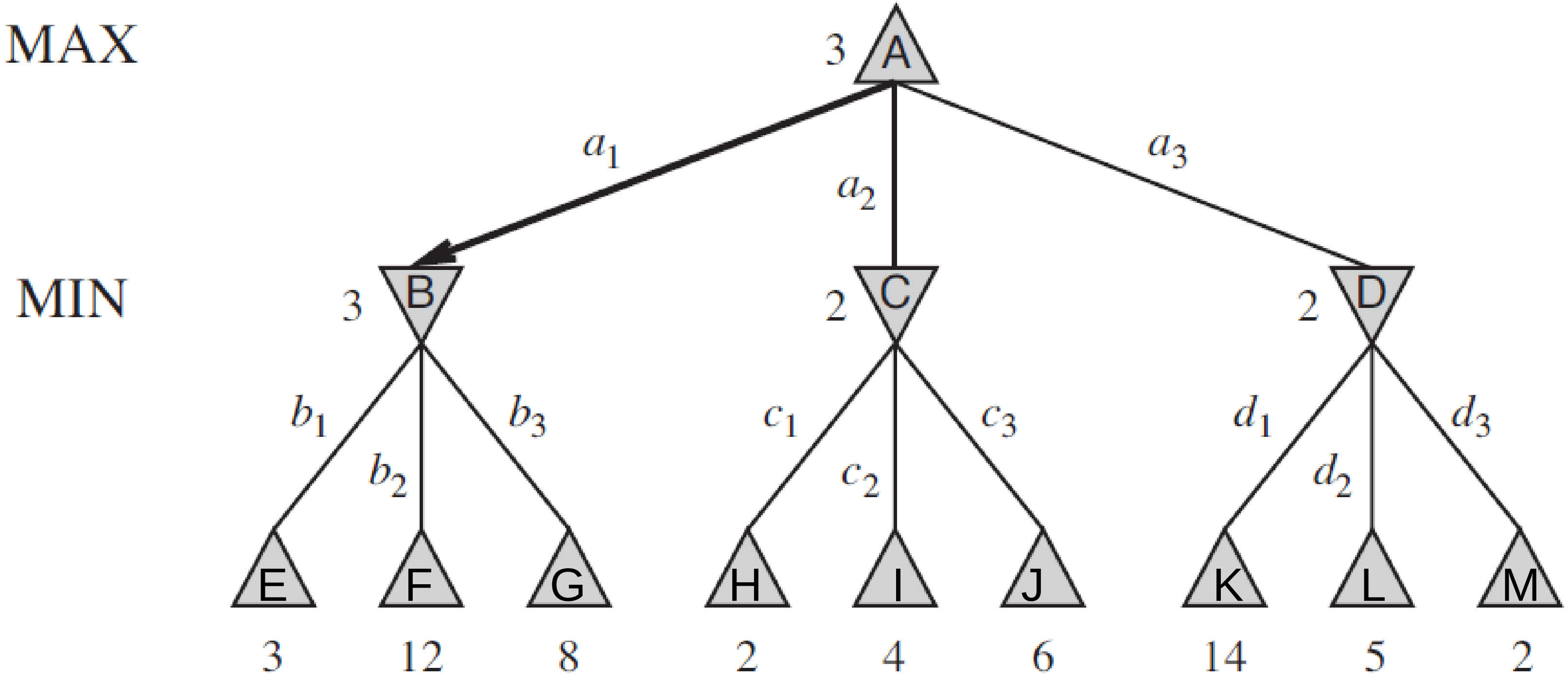
- α is the value of the best choice (found so far at any choice point along the path) for MAX
  - If (v) is worst than (alpha), MAX will avoid it
    - i.e., prune the branch
- Beta is defined similarly for MIN



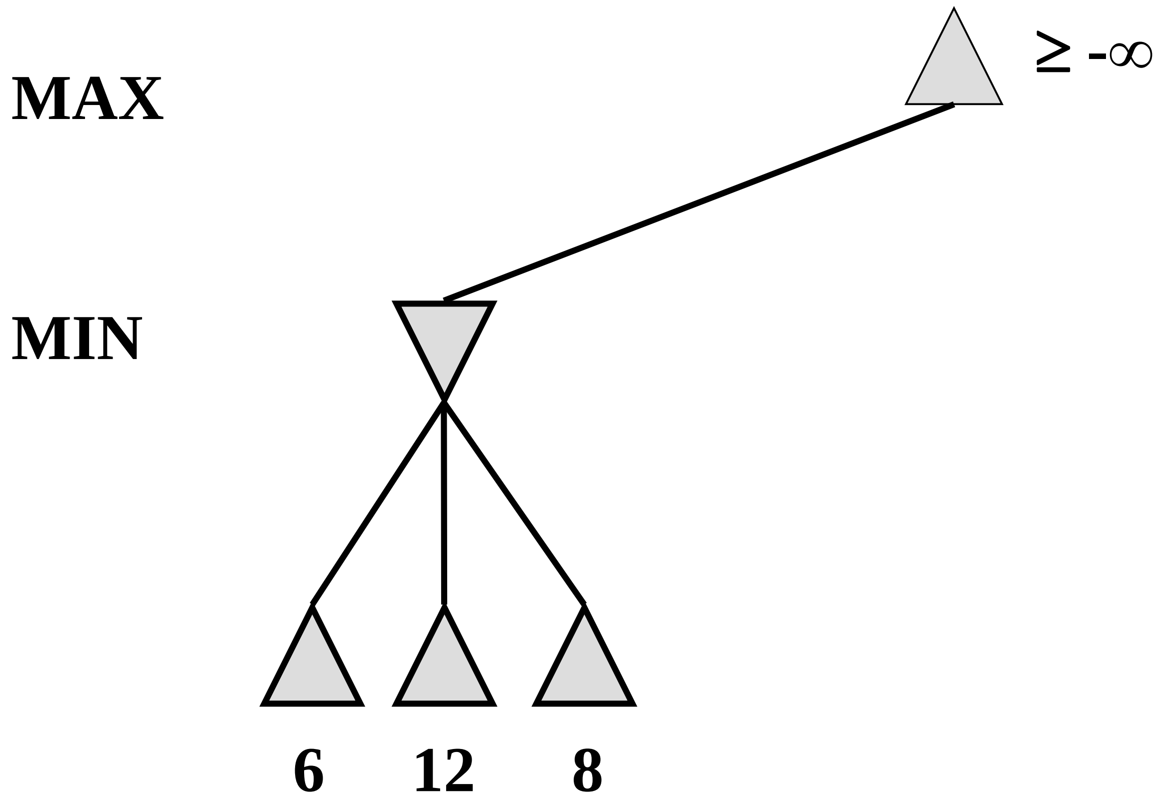
What's the worst-case move order?



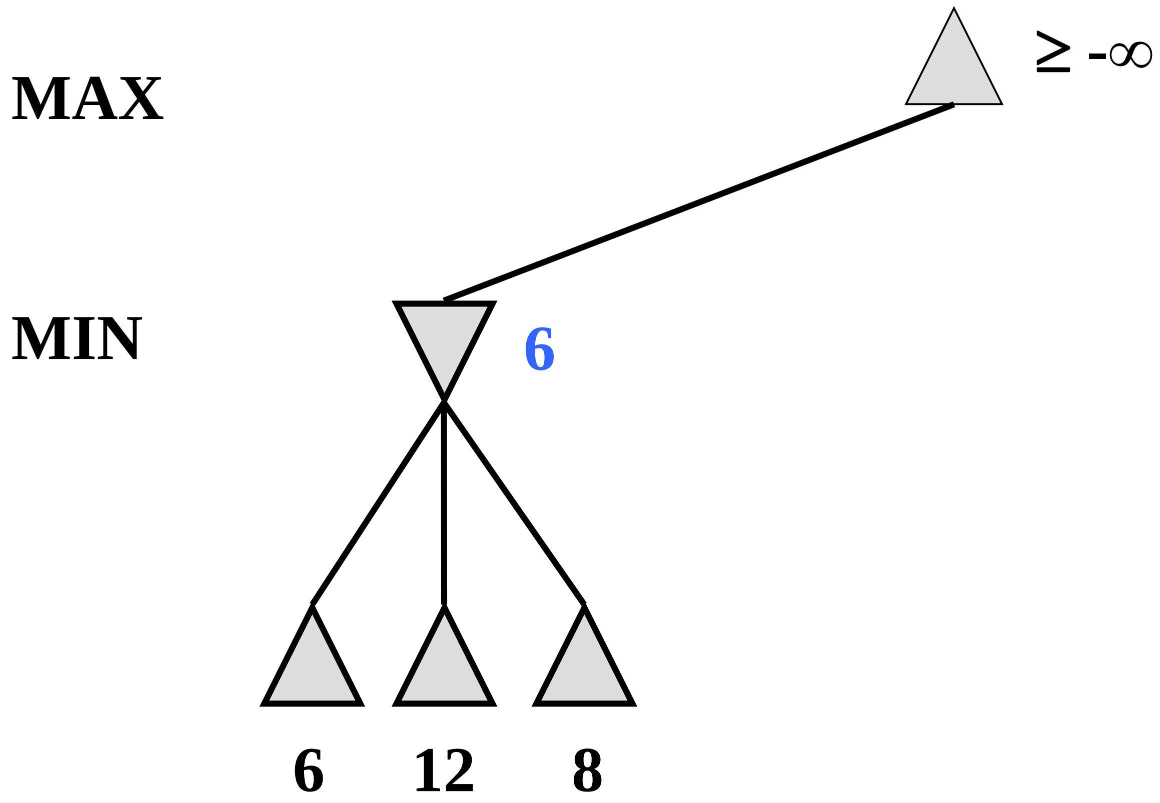
What's the best-case move order?



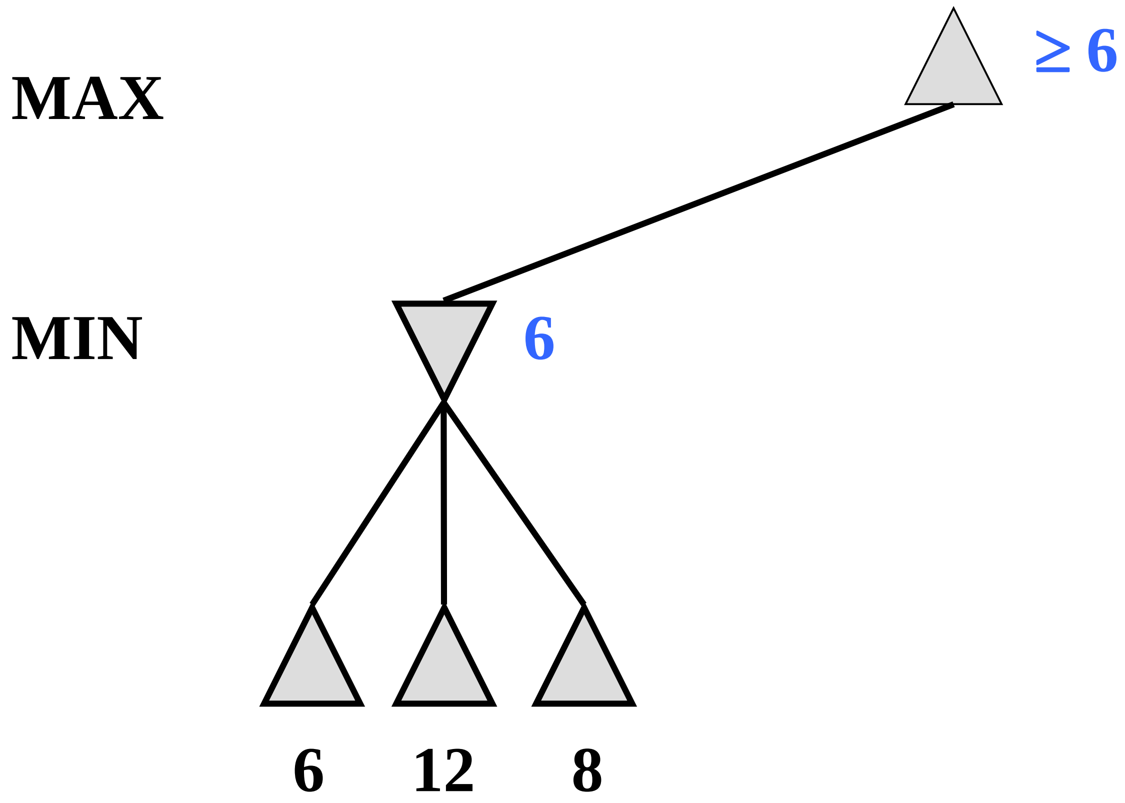
α-β pruning: example



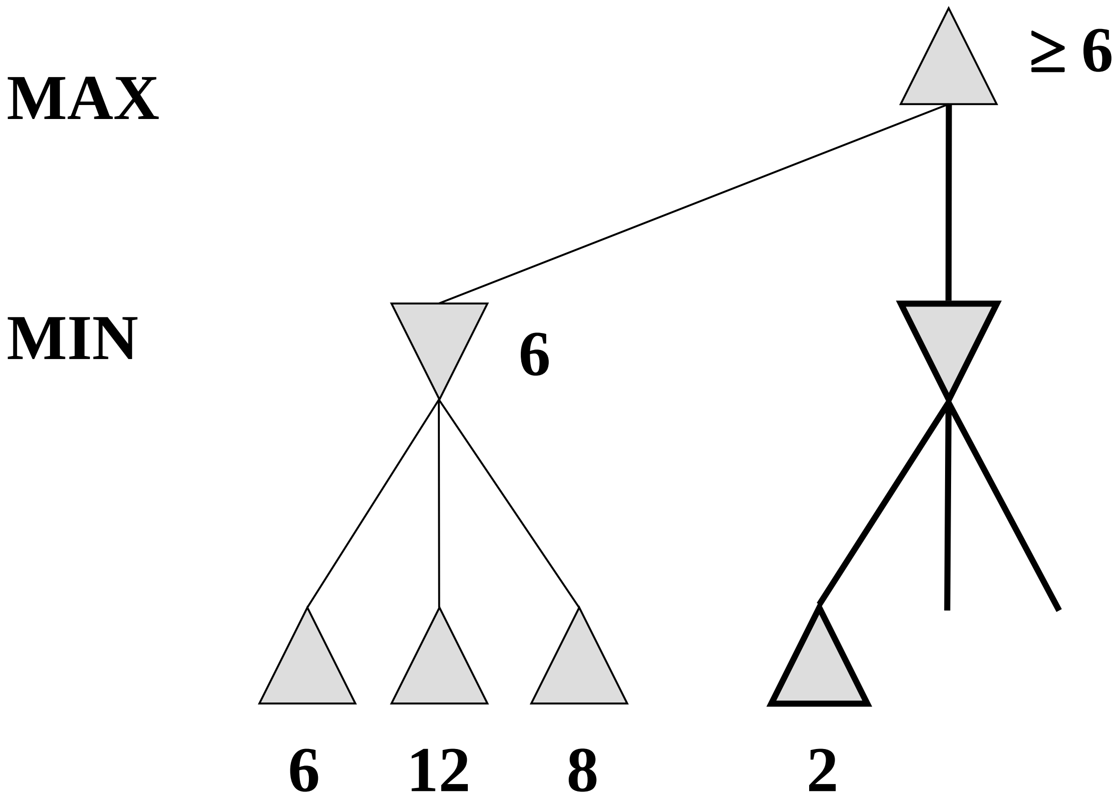
α-β pruning: example



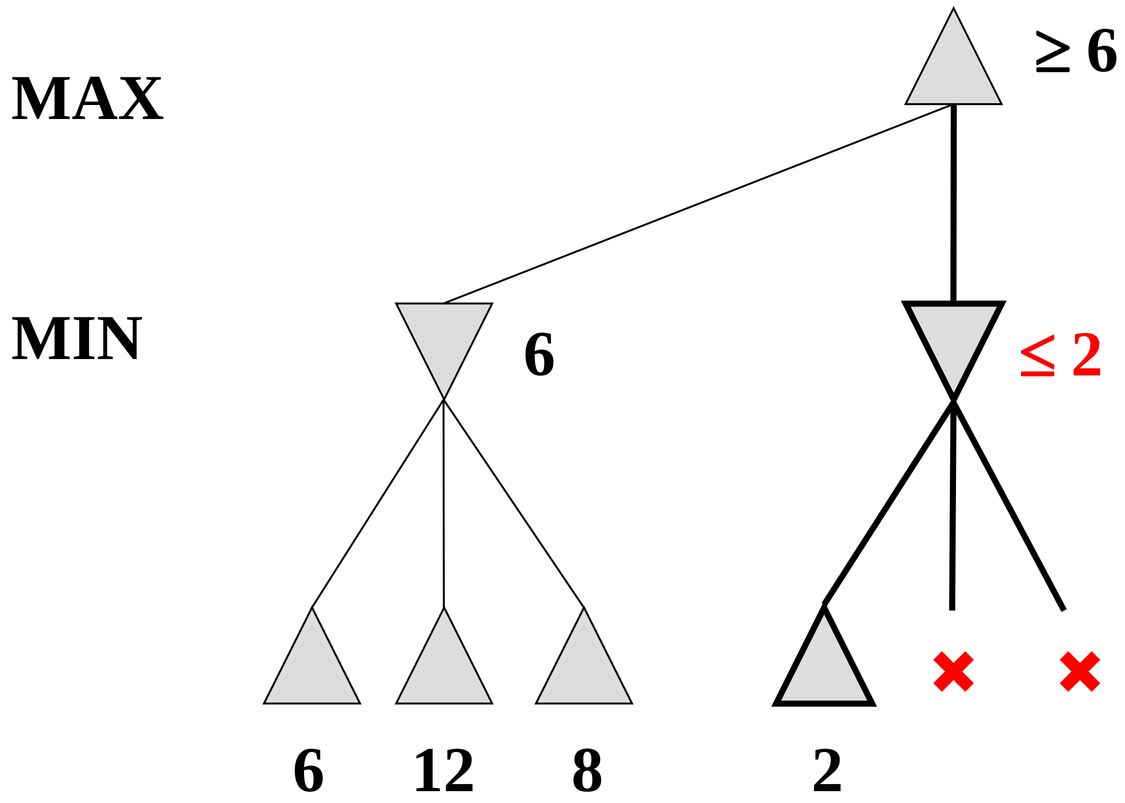
α-β pruning: example



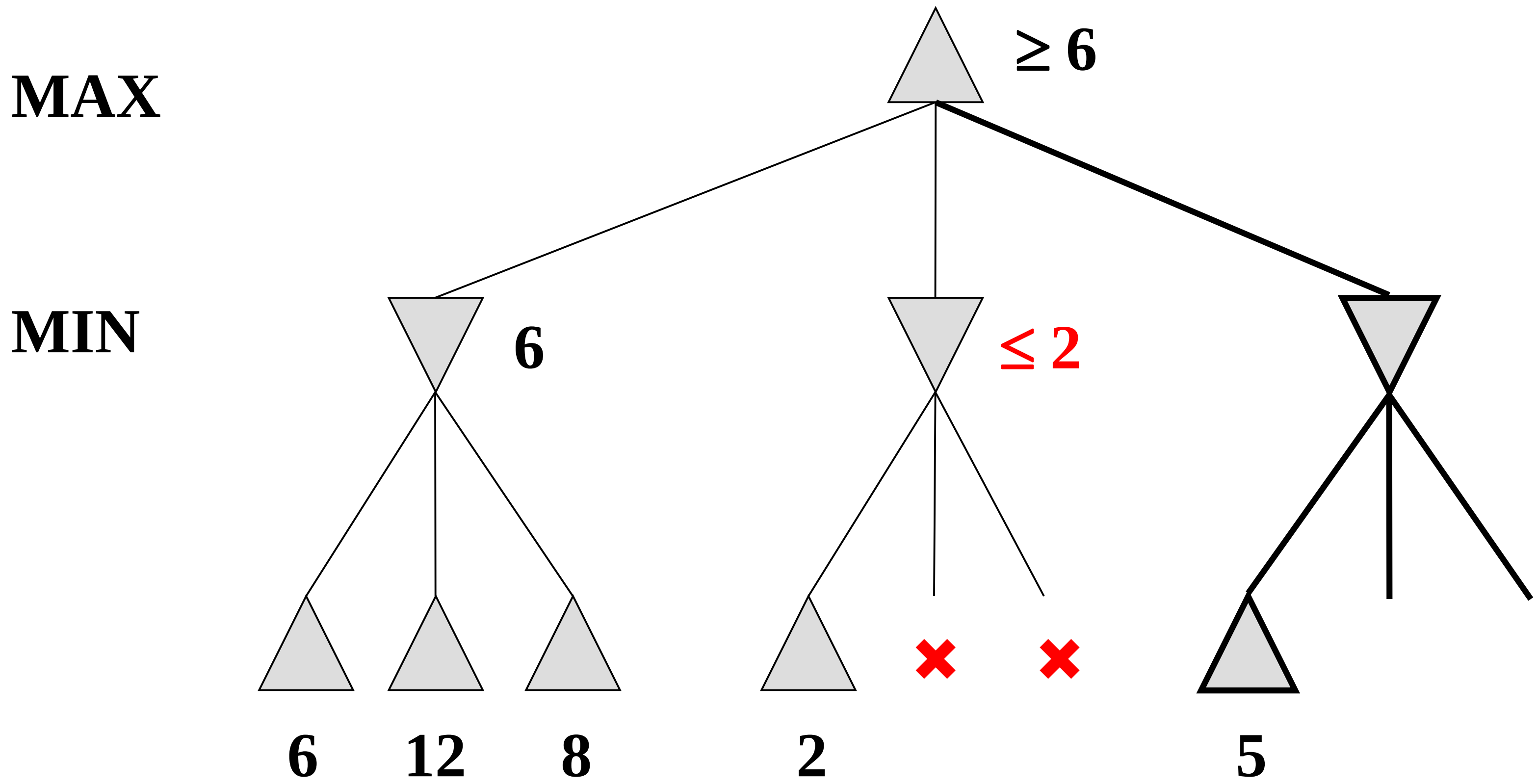
α-β pruning: example



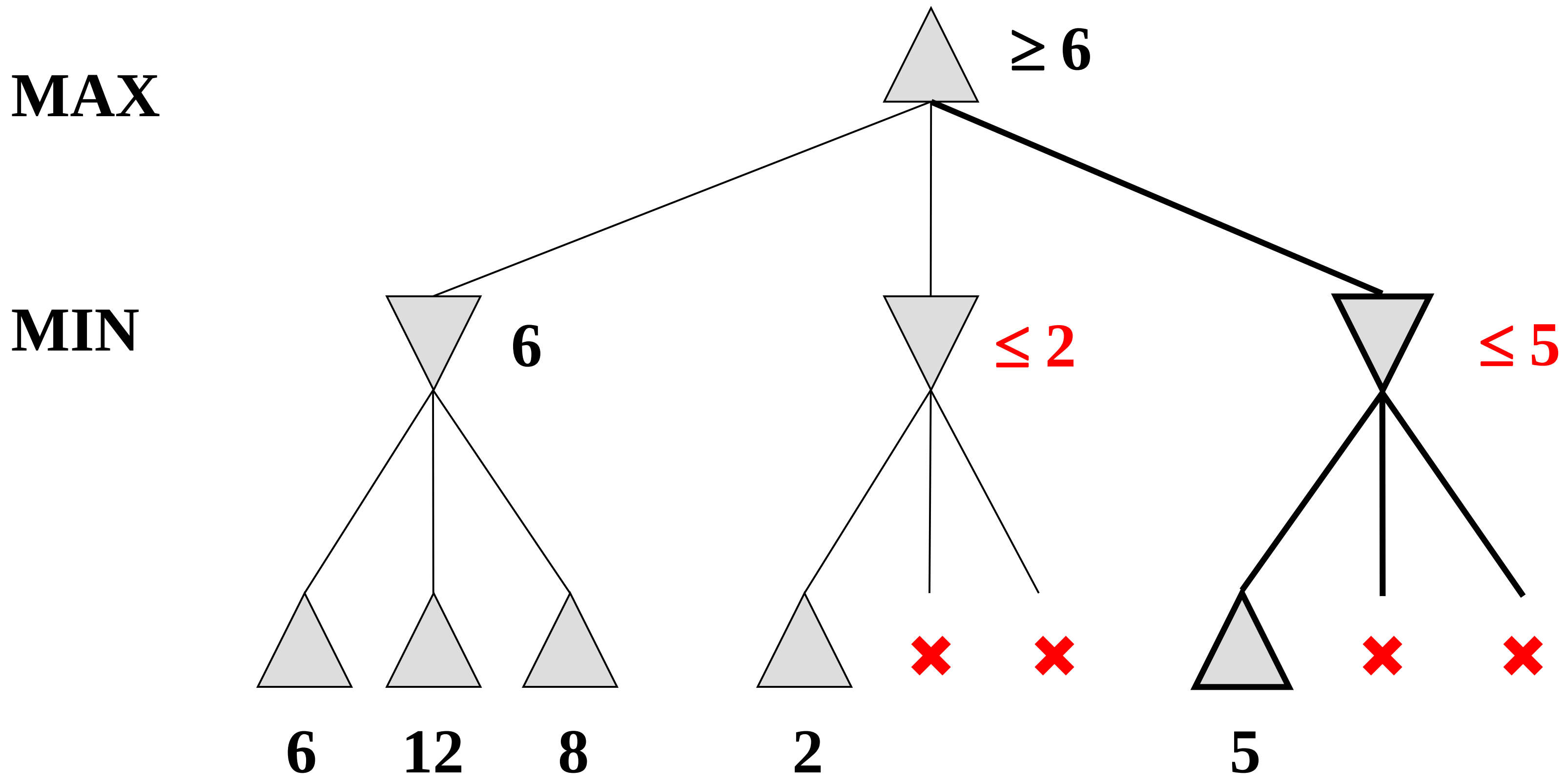
α-β pruning: example



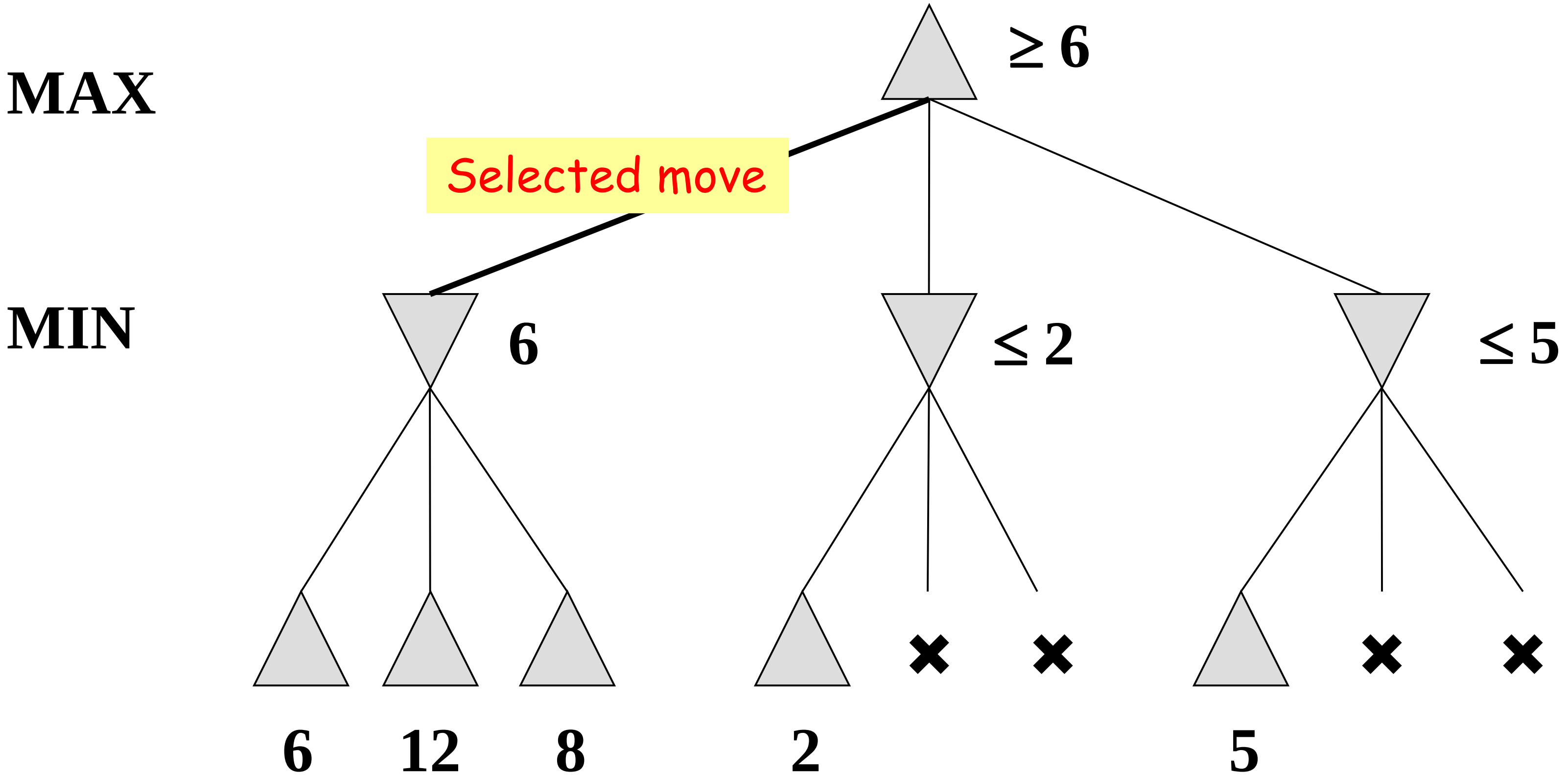
α-β pruning: example



α-β pruning: example



α-β pruning: example



Imperfect decision

- Minimax generate the **entire game search space**
- Alpha-beta pruning can **avoid large parts** of it, but still has to search all the way to at least some **terminal** states.

- Problem: What if **terminate** states are too deep to reach?

Imperfect decision

- Minimax generate the **entire game search space**
- Alpha-beta pruning can **avoid large parts** of it, but still has to search all the way to at least some **terminal** states.

- Problem: What if **terminate** states are too deep to reach?

Claude Shannon (1950): Programming a Computer for Playing Chess

Imperfect decision

Instead of using the "**Utility**" function of **a terminal state**

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

Using the "**Evaluation**" function of **a cut-off state**

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s,d) \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN}. \end{cases}$$

Etimate the expected utility of the game from state (s)

** this is what human players do, e.g. using "material values" of pieces

each pawn is worth 1, a knight or bishop is worth 3, a rook 5, and the queen 9.

Minimax with cutoff: viable algorithm?

MinimaxCutoff is identical to MinimaxValue except

1. Terminal? is replaced by Cutoff?
2. Utility is replaced by Eval

Does it work in practice?

$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

4-ply lookahead is a hopeless chess player!

4-ply ≈ human novice
8-ply ≈ typical PC, human master
12-ply ≈ Deep Blue, Kasparov

Assume we have 100 seconds, evaluate 10^4 nodes/s; can evaluate 10^6 nodes/move

Outline

- What is AI?
- Problem-solving agent
- Uninformed search
- Informed search (A*)
- Local search
- **Adversarial search**
  - The minimax algorithm
  - Alpha-beta pruning