COSC76/276 Artificial Intelligence
Fall 2022
Adversarial search

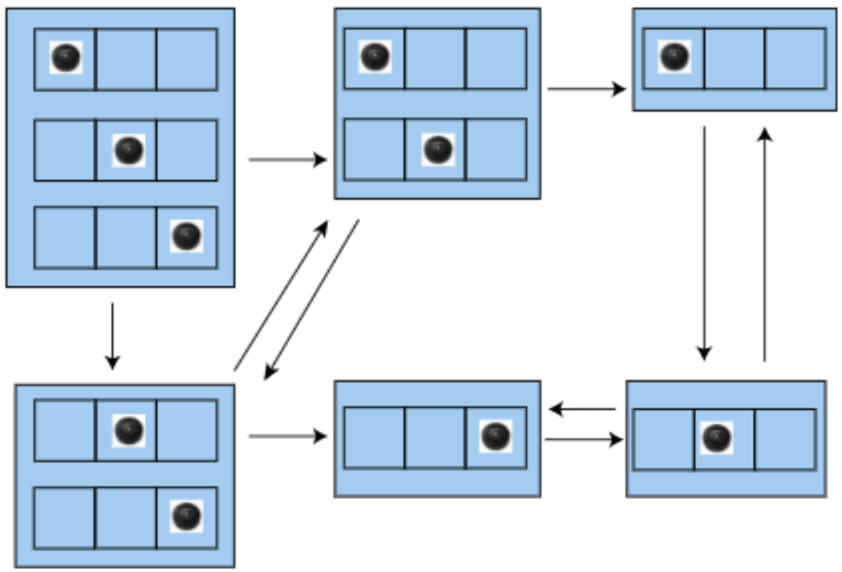
Soroush Vosoughi
Computer Science
Dartmouth College
Soroush@Dartmouth.edu

Logistics

- PA-2 due Oct 10th at 11:59pm ET
- Assignments are being steadily graded and made available
- 3 extra late days has been given

Recap: Partial observations

- Belief states: where the agent thinks it might be



- Upper bound of belief state space size: 2^3-1

Recap: Adversarial search and minimax

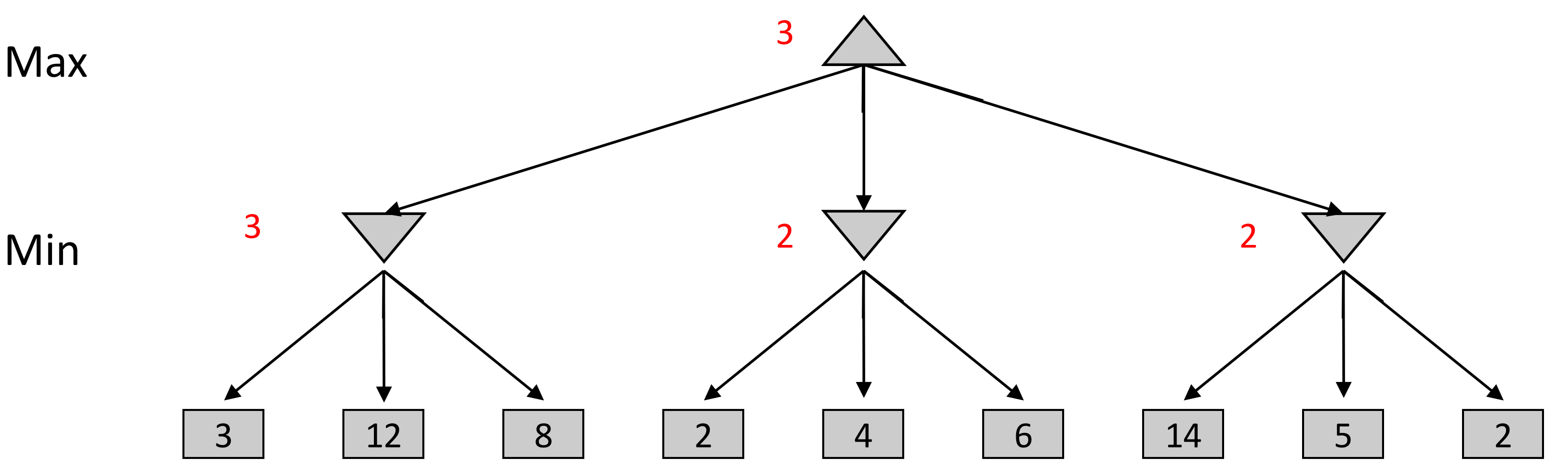
- Adversarial search as a tree search
- Minimax algorithm to find optimal strategy against optimal opponent

Minimax Example

```
def value(state):
        if the state is a terminal state: return the state's utility
        if the next agent is MAX: return max-value(state)
        if the next agent is MIN: return min-value(state)
```

```
def max-value(state):
        initialize v = -∞
        for each successor of state:
                v = max(v, value(successor))
        return v
```

```
def min-value(state):
        initialize v = +∞
        for each successor of state:
                v = min(v, value(successor))
        return v
```



Today's learning objectives

- Implement solutions that can practically work for real-world adversarial search

Outline

- Alpha-beta pruning
- Handling resource limits

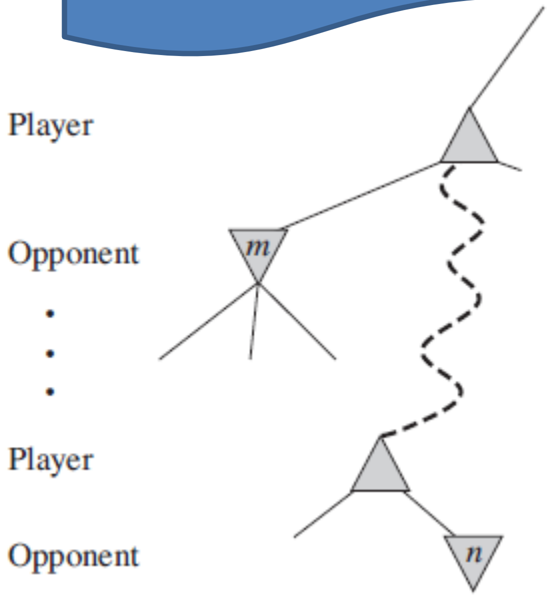
Outline

- Alpha-beta pruning
- Handling resource limits

Alpha-beta pruning intuition

- Consider a node n somewhere in the tree such that Player has a choice of moving to that node

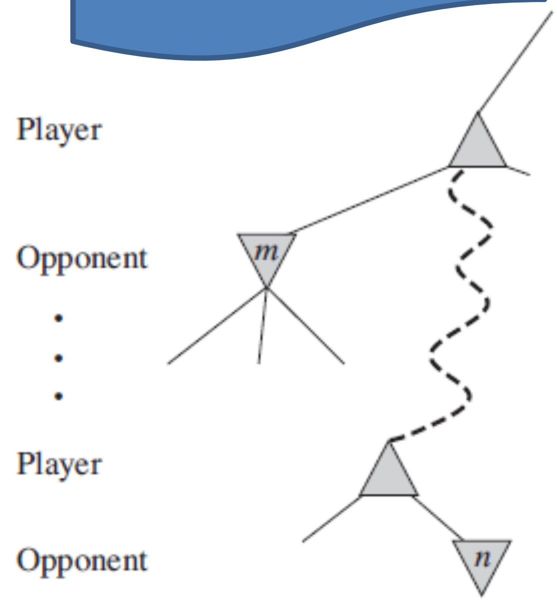
If we get n, which is worse than m, could we stop the search?



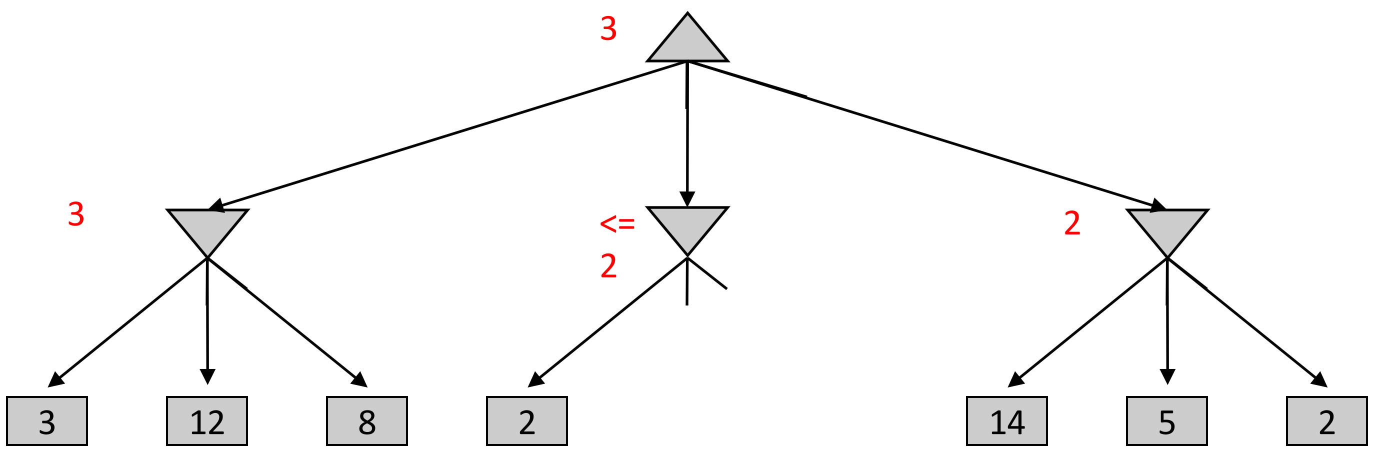
Alpha-beta pruning intuition

If we get n, which is worse than m, could we stop the search?

- Consider a node n somewhere in the tree such that Player has a choice of moving to that node
  - 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



Alpha-beta pruning intuition example



Alpha beta pruning

- Alpha: MAX's best option on path to root
- Beta: MIN's best option on path to root

```
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
```

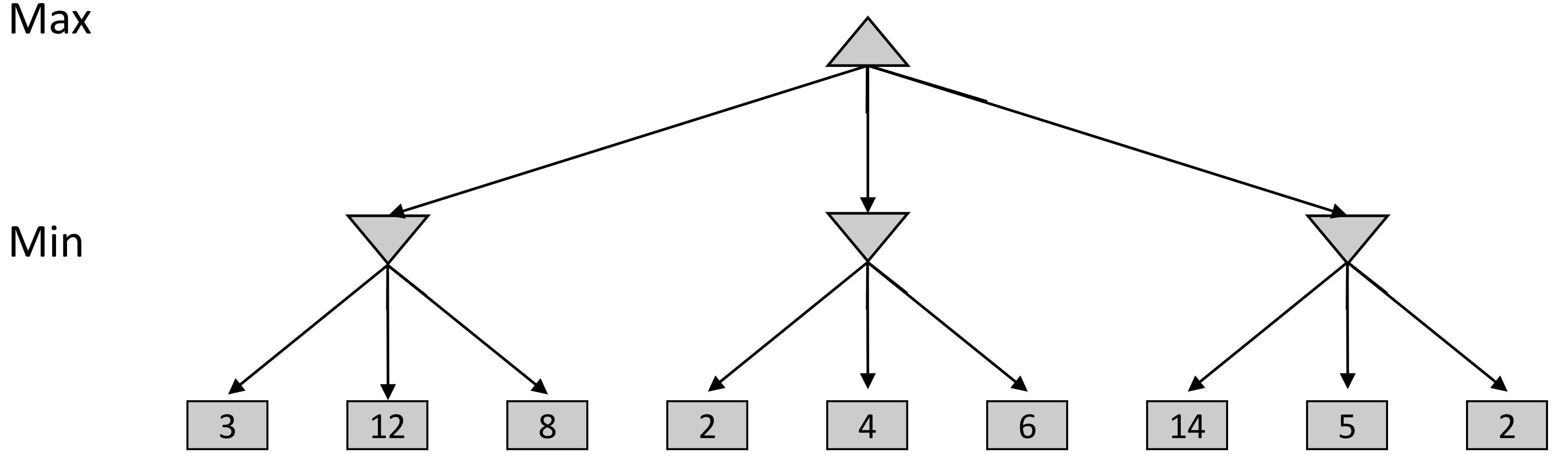
```
function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Example

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*, α, β) **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*, α, β) **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



Let's work through this example together with a full run of the algorithm

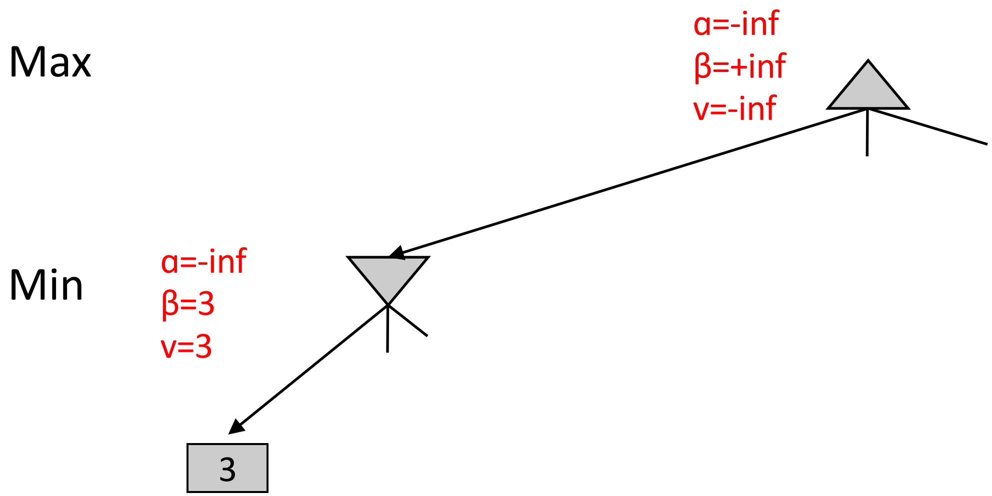
Alpha-beta pruning example following the code

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*, α, β) **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*, α, β) **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

Alpha: MAX's best option on path to root
Beta: MIN's best option on path to root



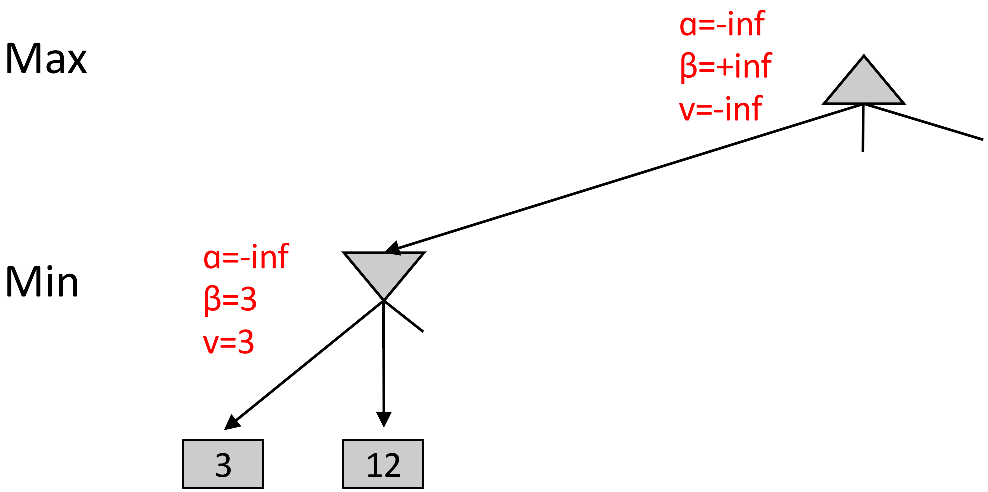
Alpha-beta pruning example following the code

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

Alpha: MAX's best option on path to root
Beta: MIN's best option on path to root

function MAX-VALUE(*state*, α, β) **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*, α, β) **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



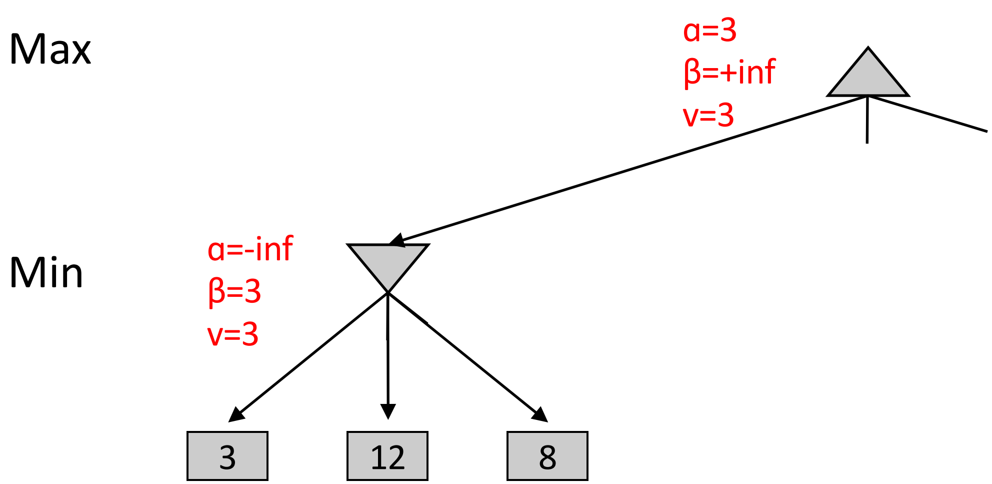
Alpha-beta pruning example following the code

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*, α, β) **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*, α, β) **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

Alpha: MAX's best option on path to root
Beta: MIN's best option on path to root



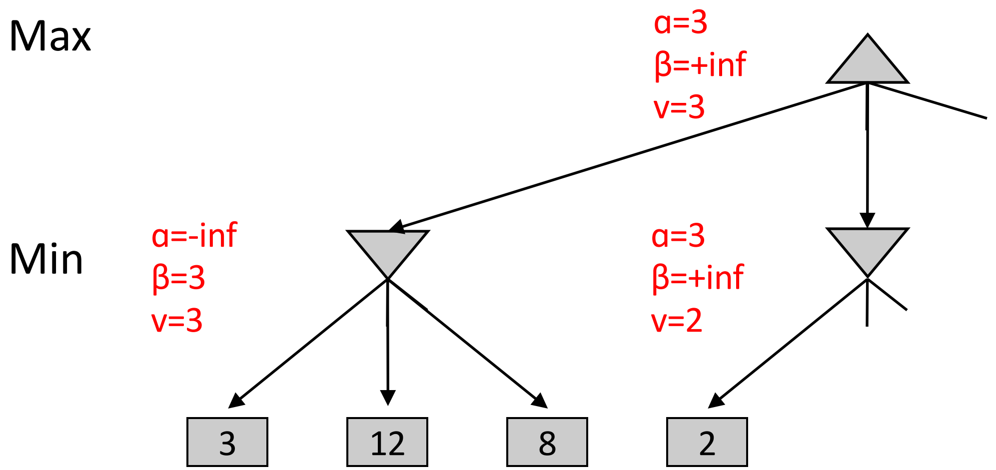
Alpha-beta pruning example following the code

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

Alpha: MAX's best option on path to root
Beta: MIN's best option on path to root

```
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
```



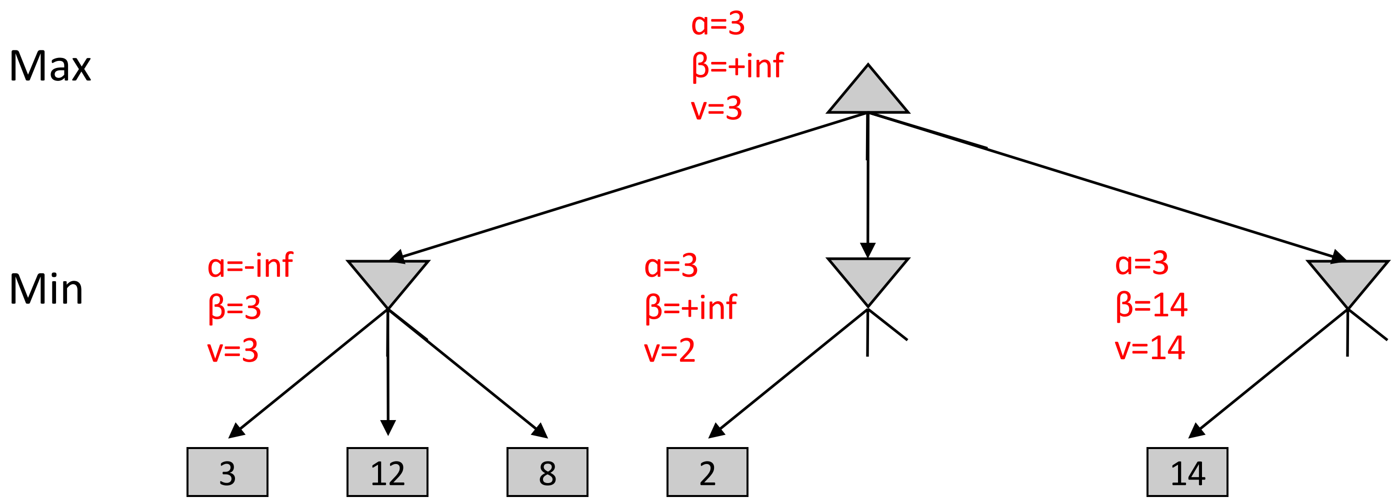
Alpha-beta pruning example following the code

```
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
```

Alpha: MAX's best option on path to root
Beta: MIN's best option on path to root



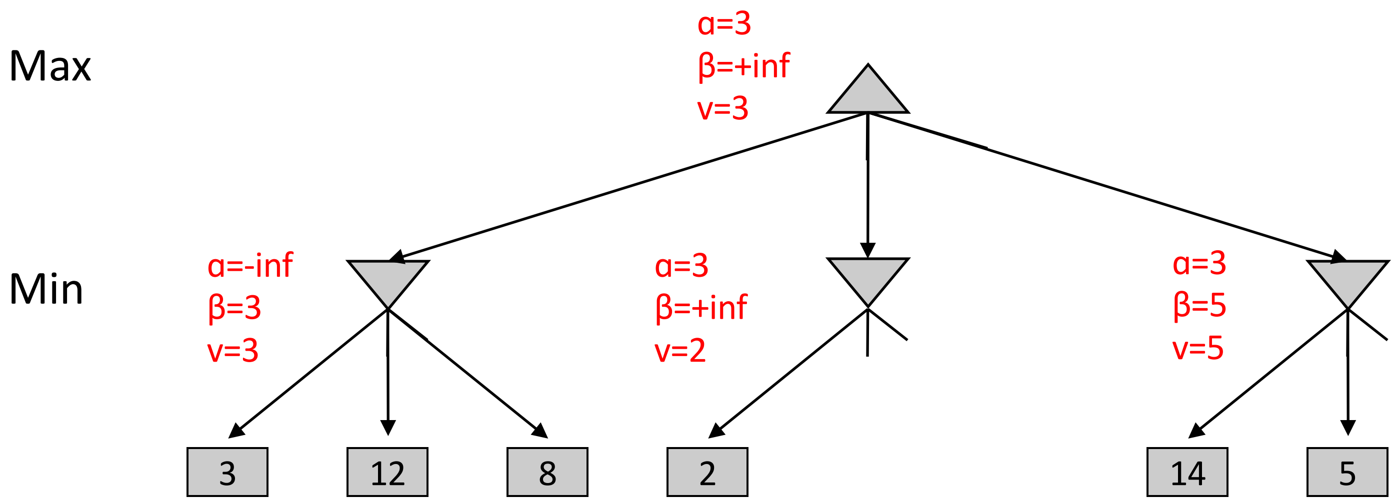
Alpha-beta pruning example following the code

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

Alpha: MAX's best option on path to root
Beta: MIN's best option on path to root

```
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
```



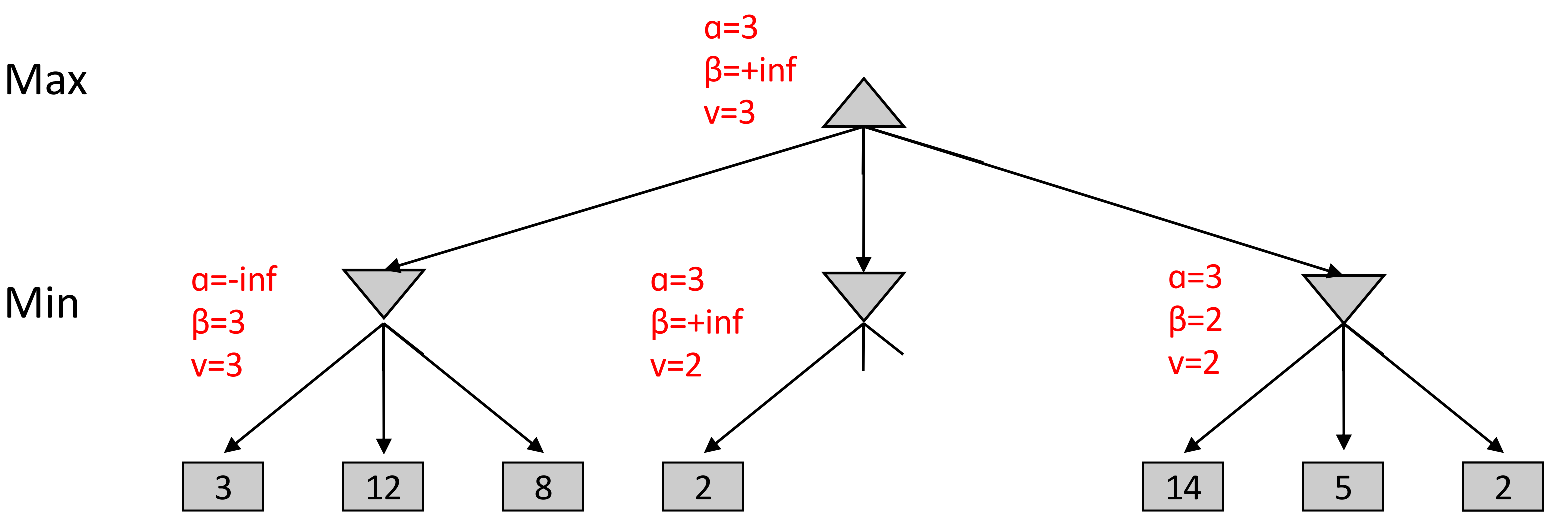
Alpha-beta pruning example following the code

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

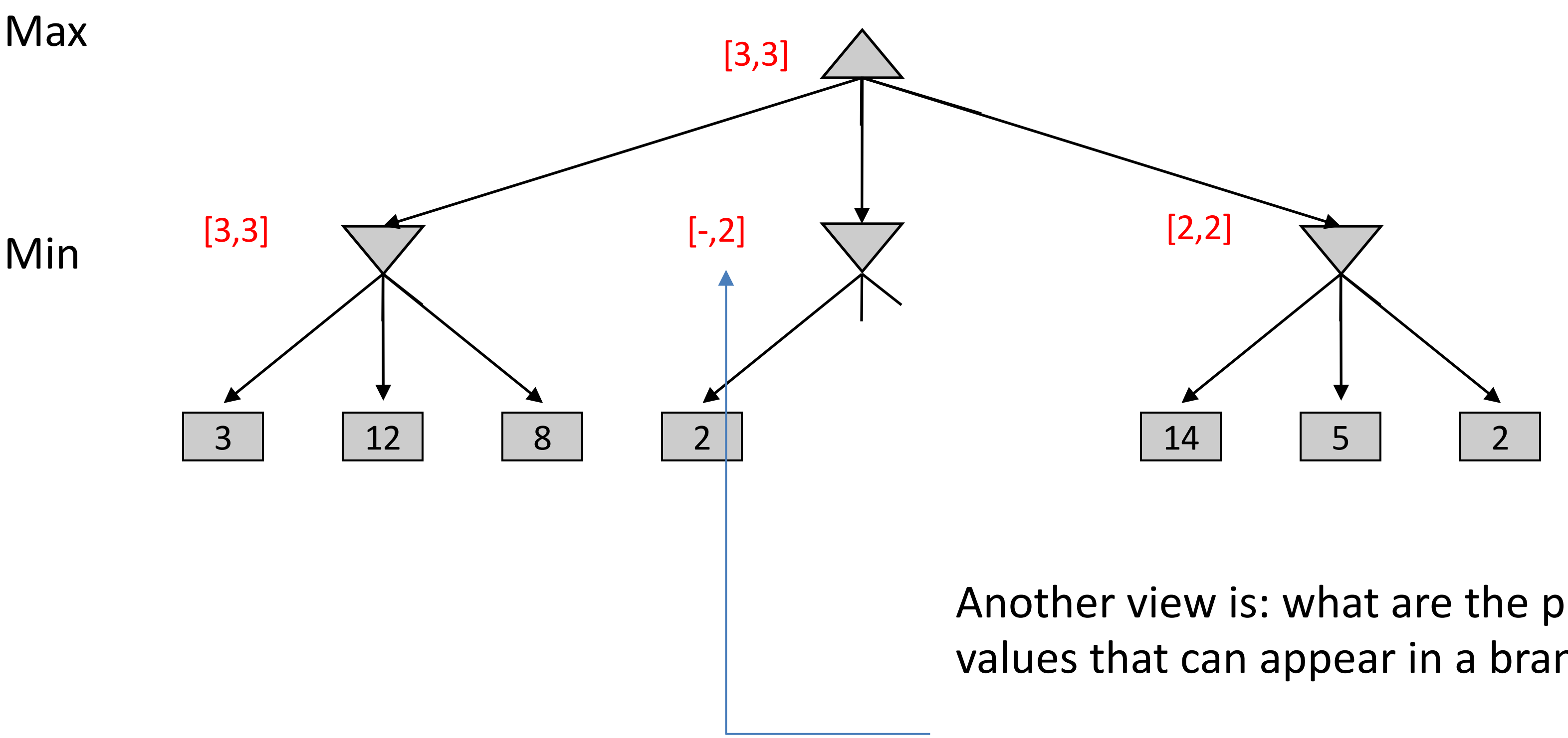
Alpha: MAX's best option on path to root
Beta: MIN's best option on path to root

function MAX-VALUE(*state*, α, β) **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*, α, β) **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



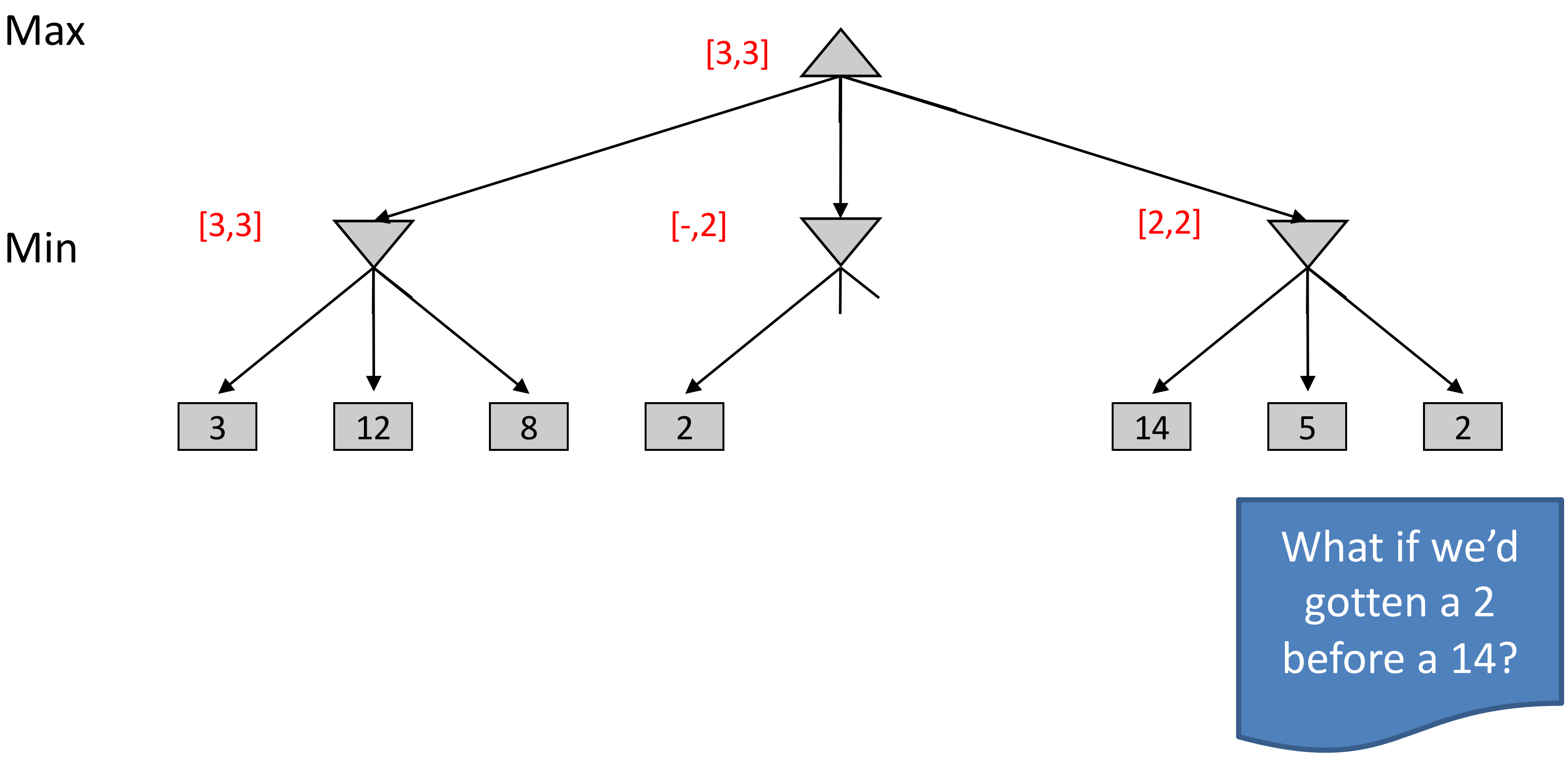
Alpha-beta pruning example



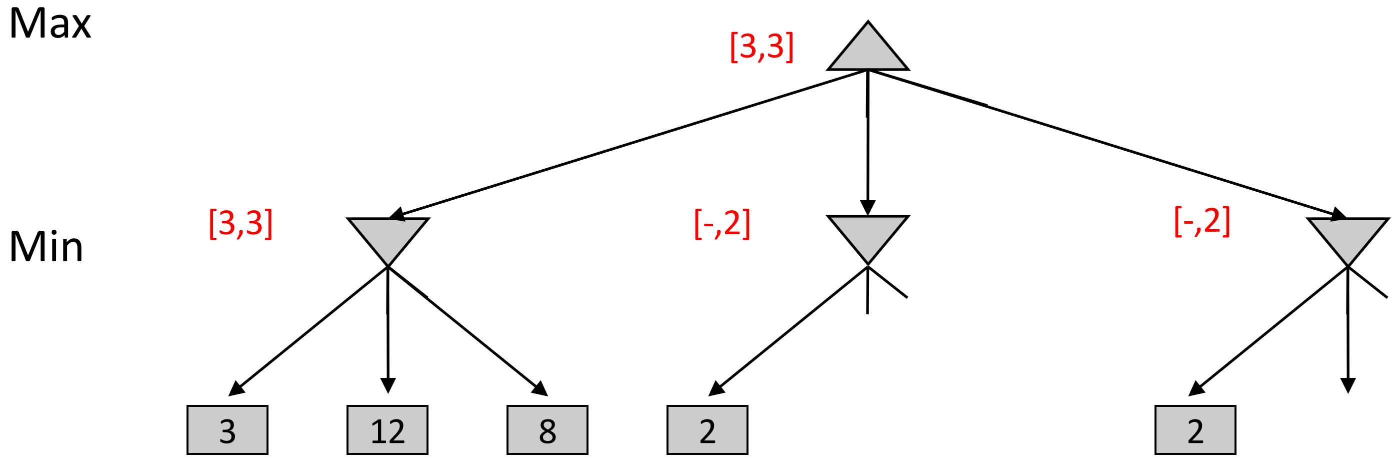
Another view is: what are the possible values that can appear in a branch?

For example, here, it could go from –inf to 2

Alpha-beta pruning example



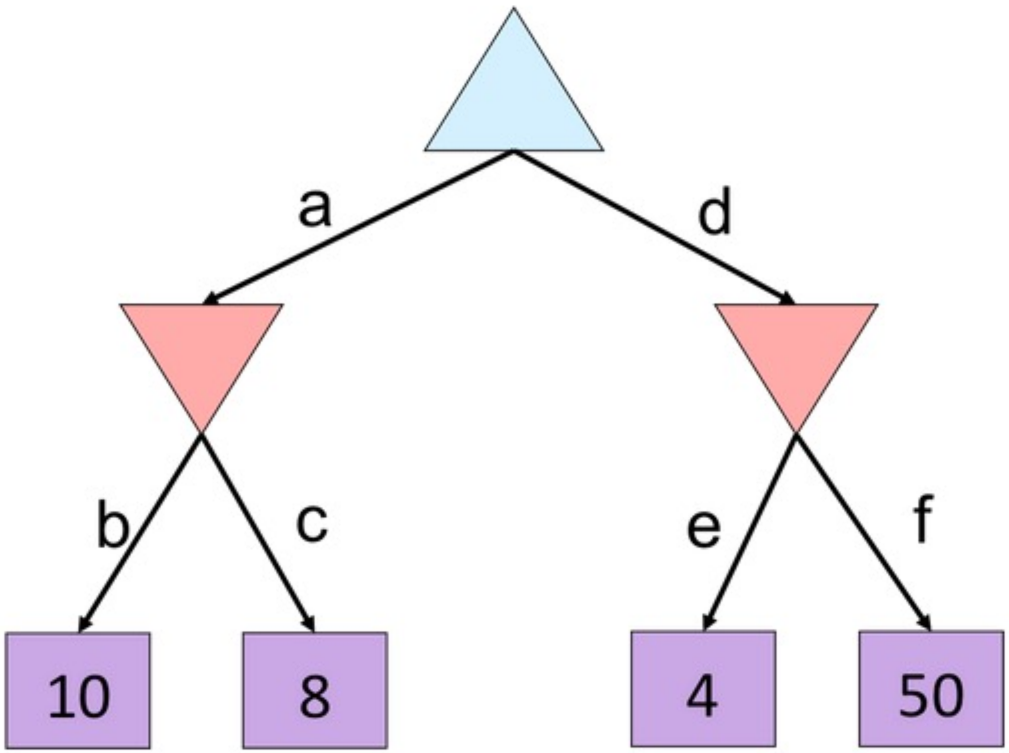
Alpha-beta pruning example



Order matters!

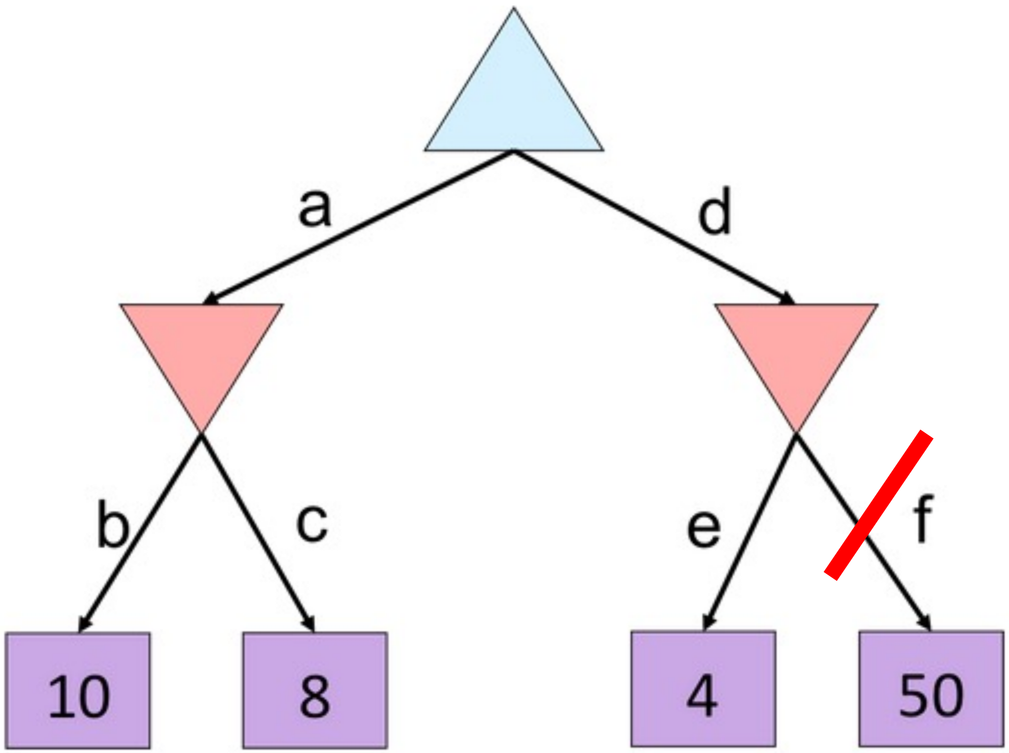
What if we'd gotten a 2 before a 14?

Alpha-Beta Quiz



Anything that can be pruned?

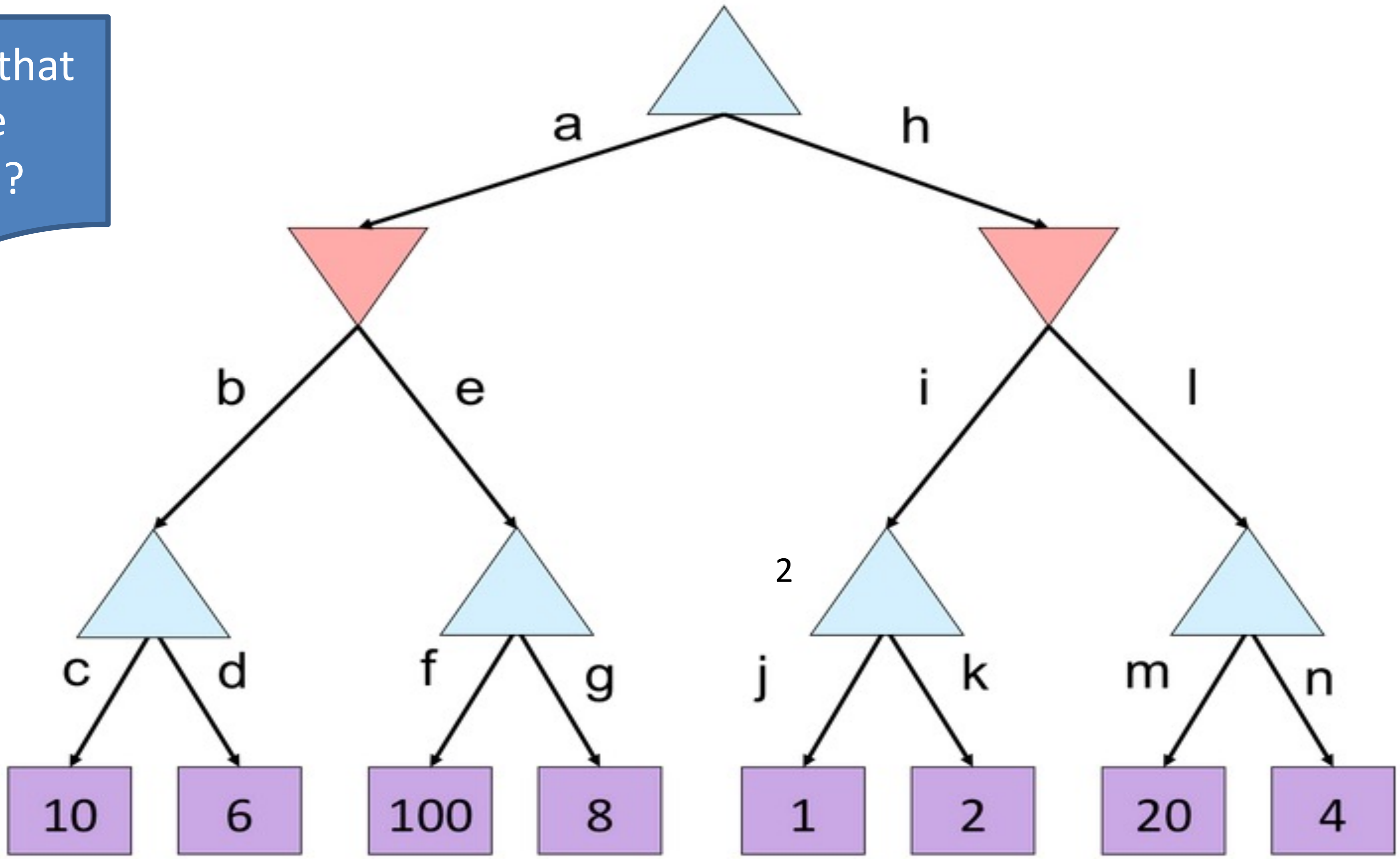
Alpha-Beta Quiz



Anything that can be pruned?

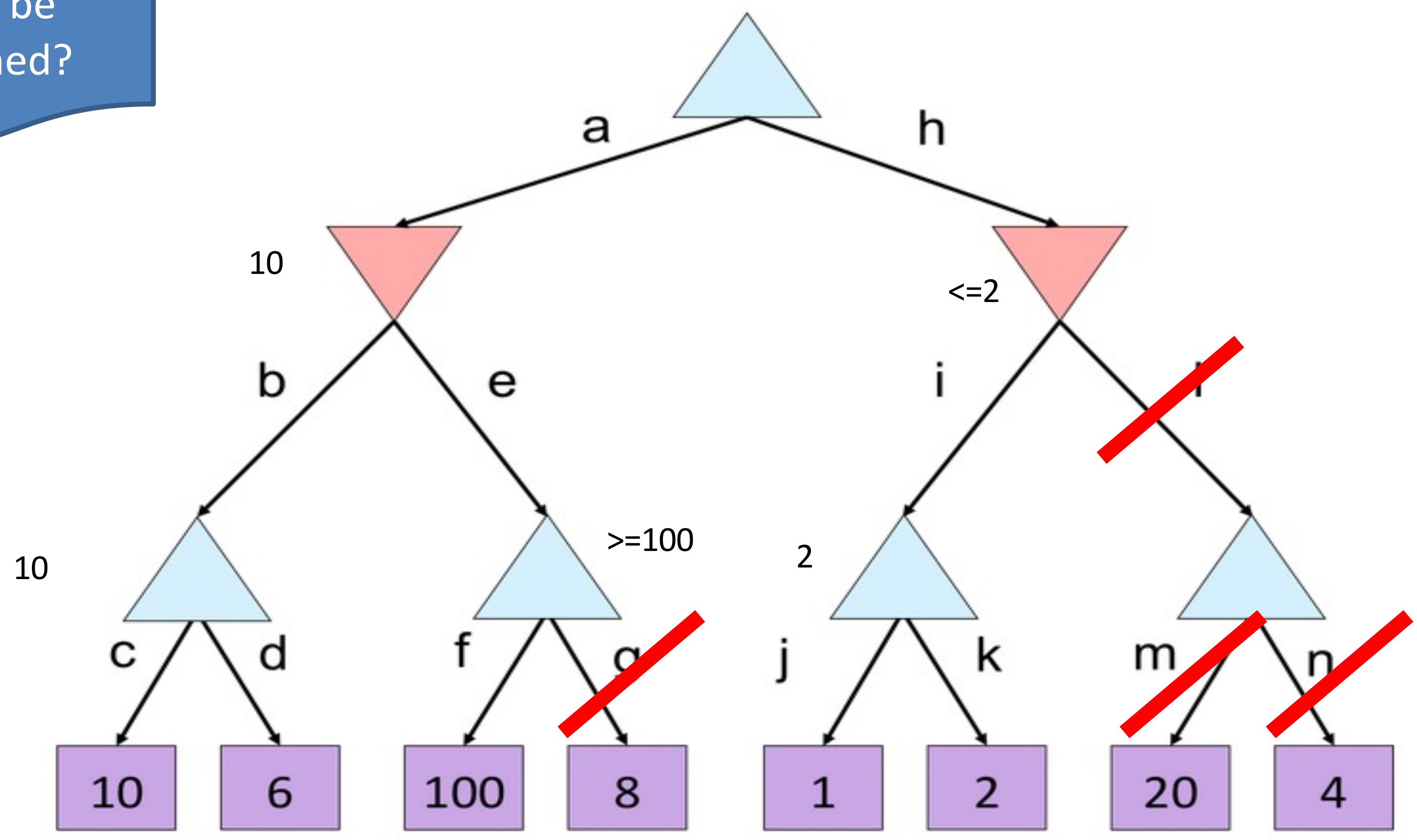
Alpha-Beta Quiz 2

Anything that can be pruned?



Alpha-Beta Quiz 2

Anything that can be pruned?



Alpha-beta pruning Properties

- This pruning has no effect on minimax value computed for the root

- Good child ordering improves effectiveness of pruning

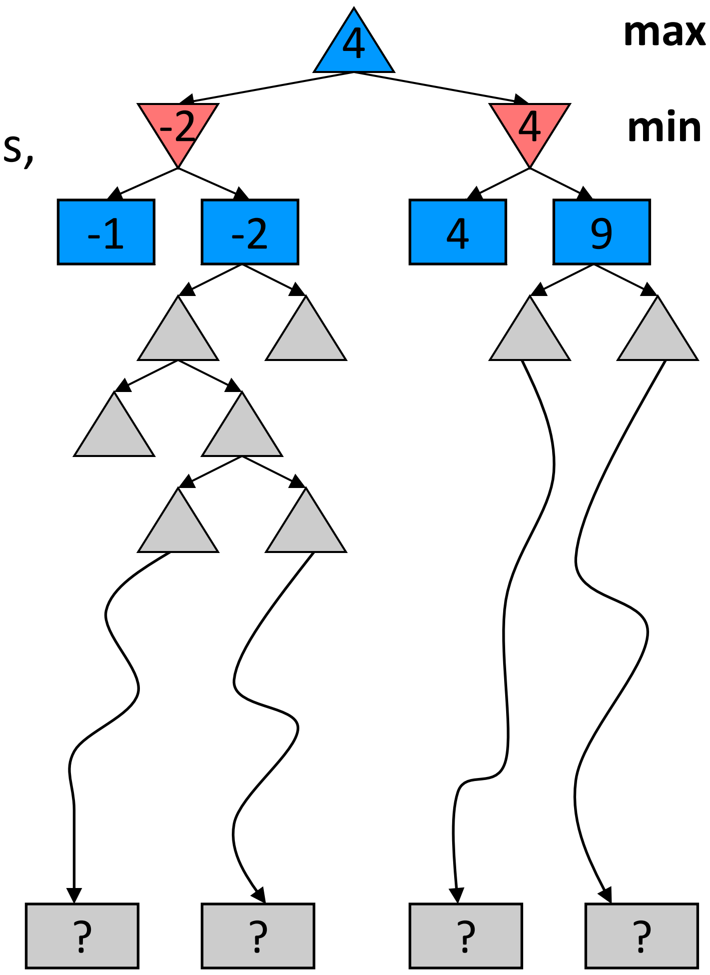
- With “perfect ordering” Time complexity drops to $O(b^{m/2})$

Outline

- Alpha-beta pruning
- Handling resource limits

Resource limits

- Problem
  - In realistic games, cannot search to leaves, as decisions need to happen in **real-time**
- Solution
  - Cut-off test instead of terminal test
    - E.g., depth-limited search, iterative deepening
  - Replace terminal utilities with an evaluation function for non-terminal positions to estimate desirability of position
- Consequence
  - Guarantee of optimal play is gone



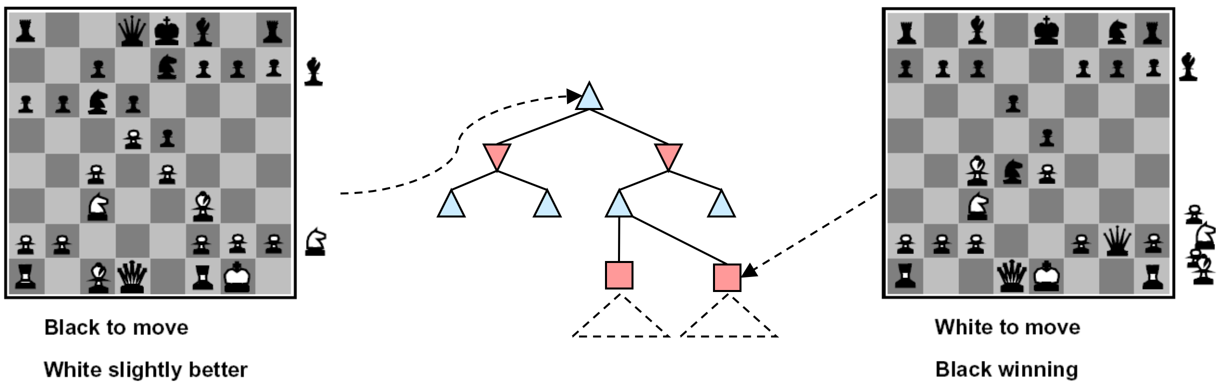
Cut-off search

- In the recursive call, we need to have bookkeeping of the depth

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

Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search



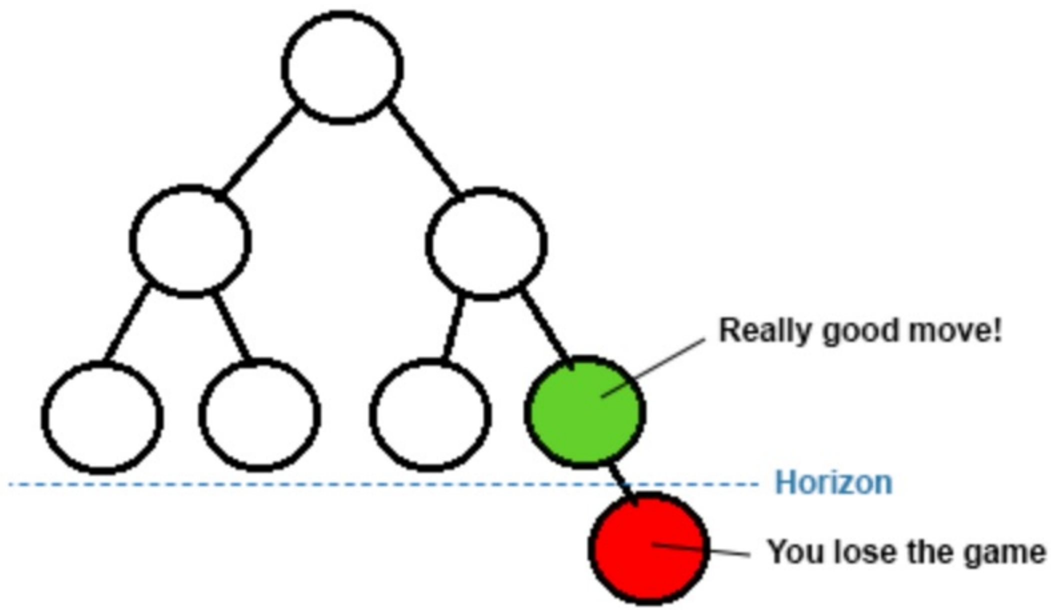
- Ideal function: returns the actual minimax value of the position
- In practice, typically weighted linear sum of features:

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

- e.g. $f_1(s)$ = (num white queens – num black queens), etc.

When to have the cut-off?

- Iterative deepening within allocated time
- **Horizon effects** are a problem: stalling tactics can push bad states beyond the depth searched



When to have the cut-off?

- Iterative deepening within allocated time
- **Horizon effects** are a problem: stalling tactics can push bad states beyond the depth searched
- **Quiescent states**, i.e., states that will not have a large change in value, are good candidates for cutoff
  - Quiescence search: search “volatile” positions to a greater depth than “stable” ones

Repeated states

- In games, repeated states occur frequently because of transpositions – i.e., different permutations of the move sequence end up in the same position
  - e.g., [a1, b1, a2, b2] vs. [a1, b2, a2, b1]
- It's worthwhile to store the evaluation of this position in a hash table – **transposition table** – the first time it is encountered
  - similar to the "explored set" in graph-search
- Tradeoff:
  - Transposition table can be too big
  - Which to keep and which to discard

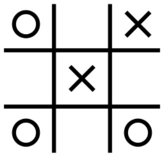
PA-3: chess game

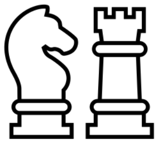
Summary

- Alpha-beta pruning
  - Alpha: MAX’s best option on path to root
  - Beta: MIN’s best option on path to root
  - Prune part of the tree that won’t be played given the current values of alpha and beta
- Real-time decisions
  - Terminal-test -> Cut-off test
  - Evaluation function to estimate desirability of a position

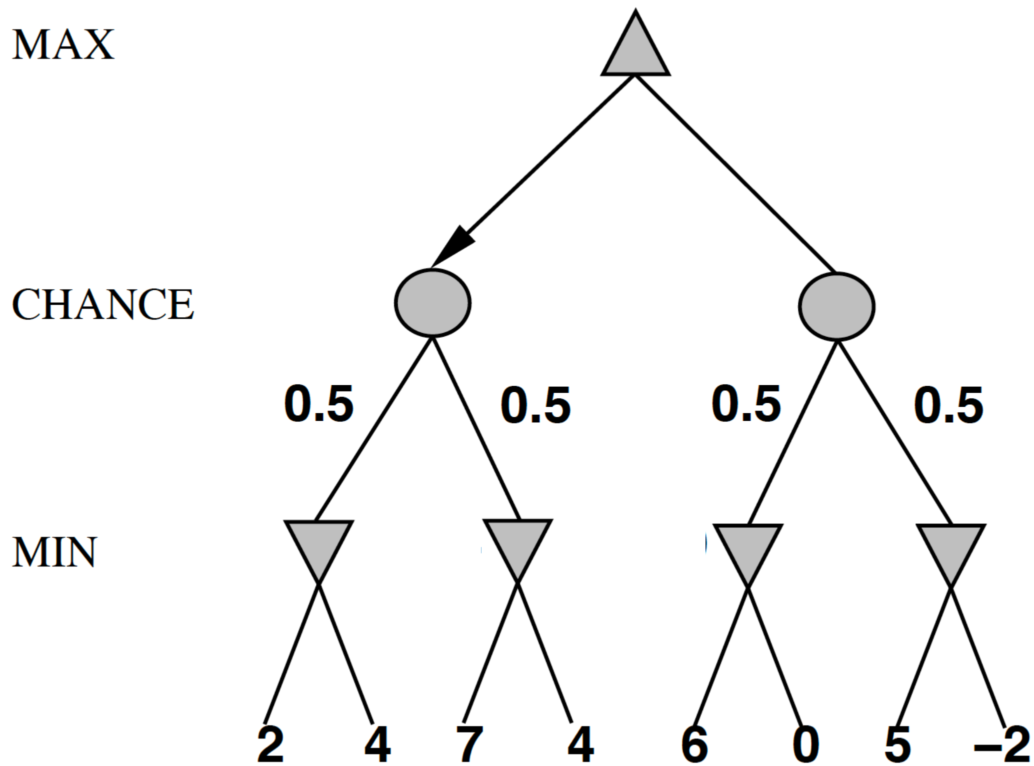
Types of Games

- Classified over different axes:
  - Number of players
  - Zero sum
  - Deterministic or stochastic
  - Perfect or imperfect information



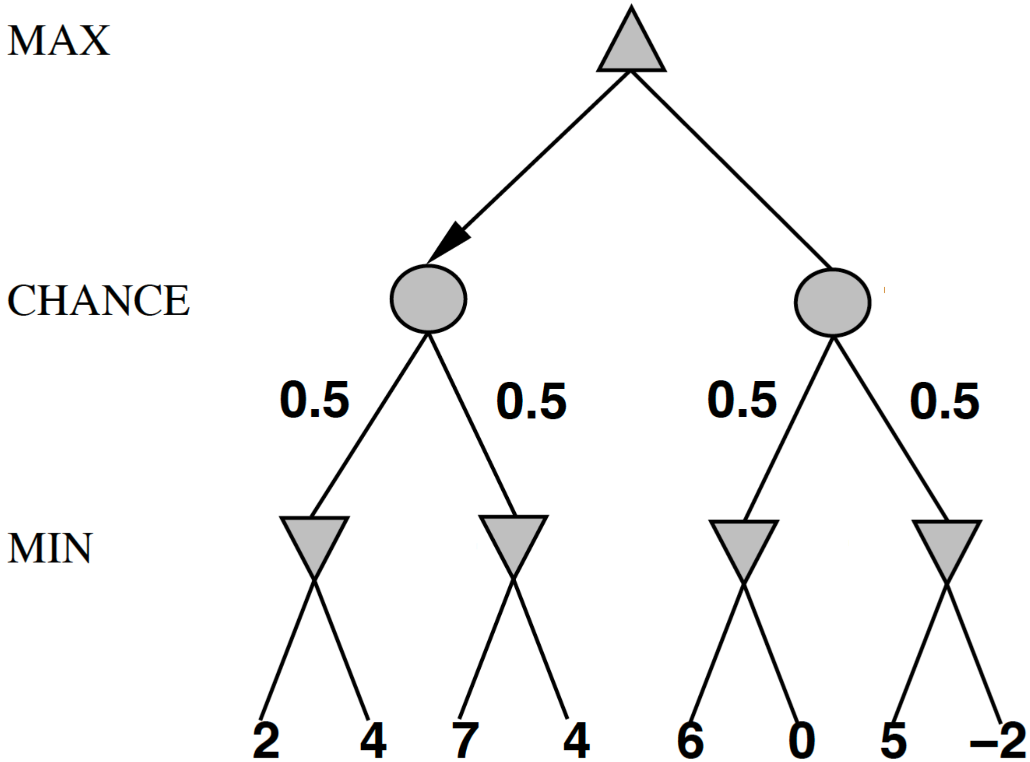


Stochastic games



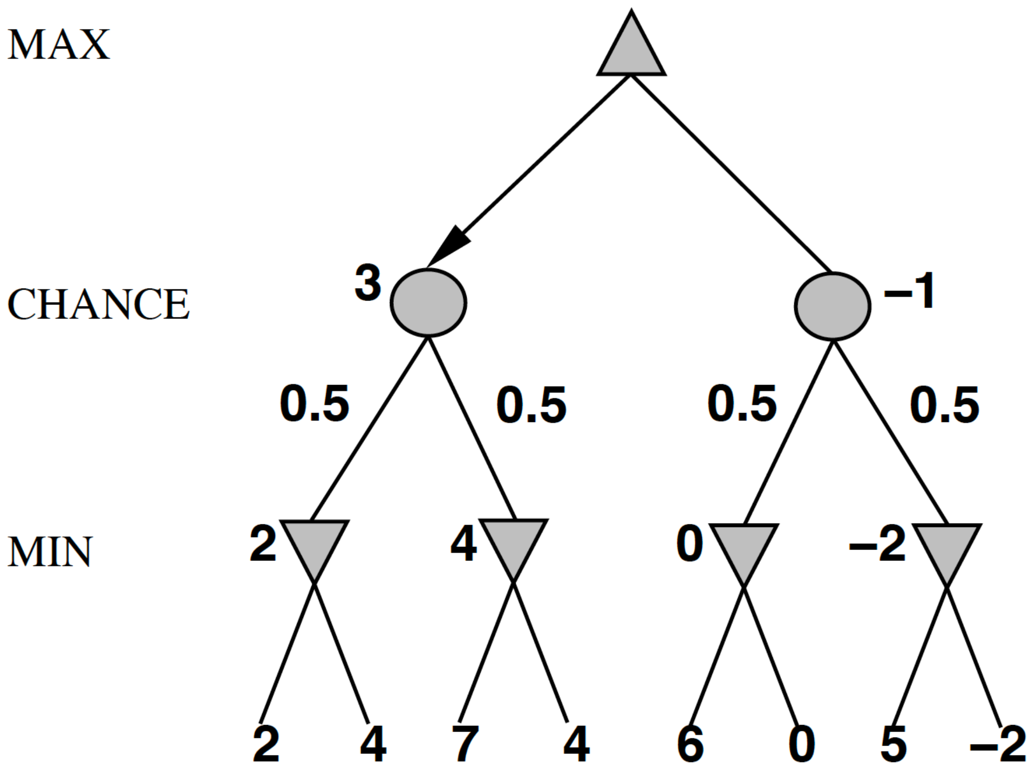
In stochastic games, uncertain outcomes controlled by chance, not an adversary (e.g., dice, card shuffling, unpredictable opponents)

Why not minimax?



Discussion

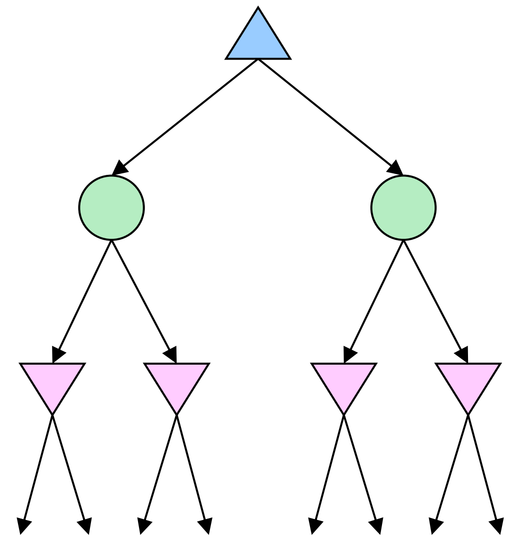
Why not minimax?



- Worst case reasoning is too conservative
- Need average case reasoning

Expectiminimax Search

- In stochastic games, values should reflect average-case (expectiminimax) outcomes, not worst-case (minimax) outcomes
- Expectiminimax search: compute the average score under optimal play
  - Max and min nodes as in minimax search
  - Chance nodes are like another player with “actions” with associated probabilities
  - Calculate their expected utilities, i.e. weighted average (expectation) of children



Expectiminimax

$$
\text{EXPECTIMINIMAX}(s) =
\begin{cases}
\text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\
\max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\
\min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\
\sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE}
\end{cases}
$$

```
def value(state):
        if the state is a terminal state: return the state's utility
        if the next agent is MAX: return max-value(state)
        if the next agent is MIN: return min-value(state)
        if the next agent is EXP: return exp-value(state)
```

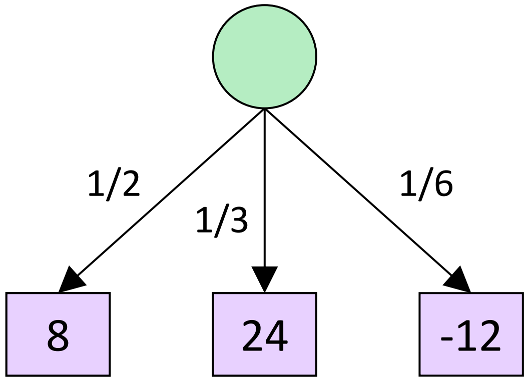
```
def max-value(state):
        initialize v = -∞
        for each successor of state:
                v = max(v, value(successor))
        return v
```

```
def min-value(state):
        initialize v = +∞
        for each successor of state:
                v = min(v, value(successor))
        return v
```

```
def exp-value(state):
        initialize v = 0
        for each successor of state:
                p = probability(successor)
                v += p * value(successor)
        return v
```

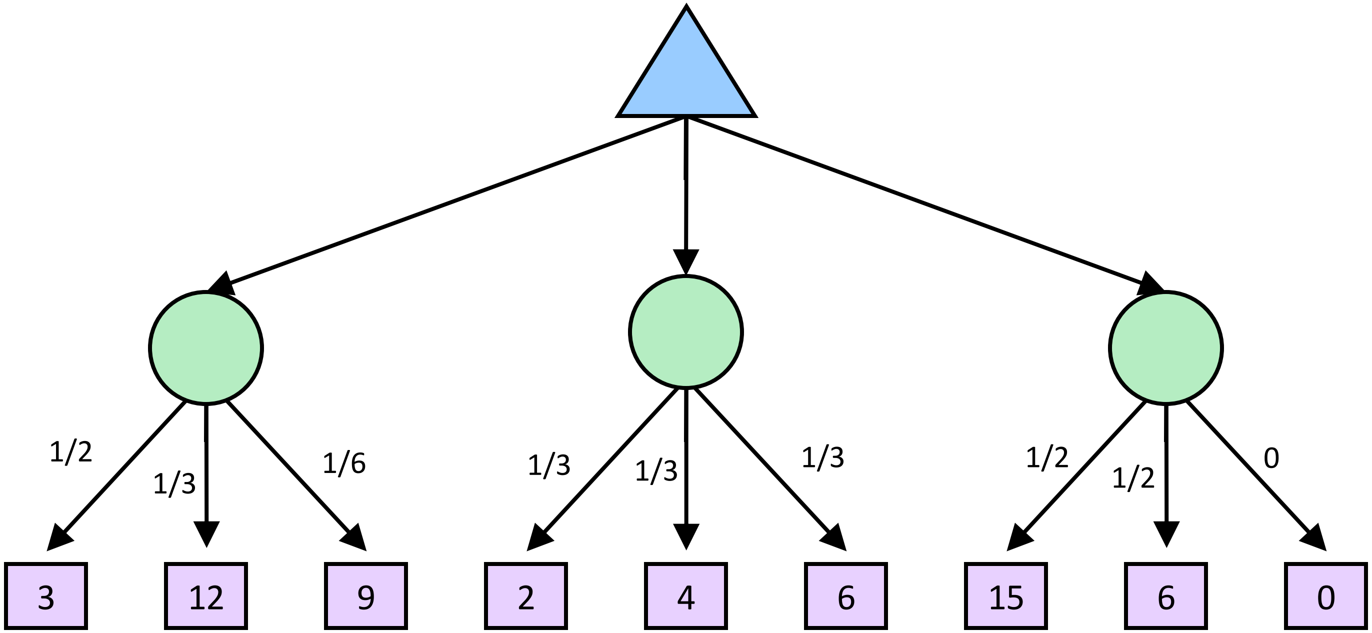
Expectimax example of chance node

$$\textsc{Expectiminimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \\ \sum_r P(r)\textsc{Expectiminimax}(\textsc{Result}(s,r)) & \text{if } \textsc{Player}(s) = \textsc{chance} \end{cases}$$



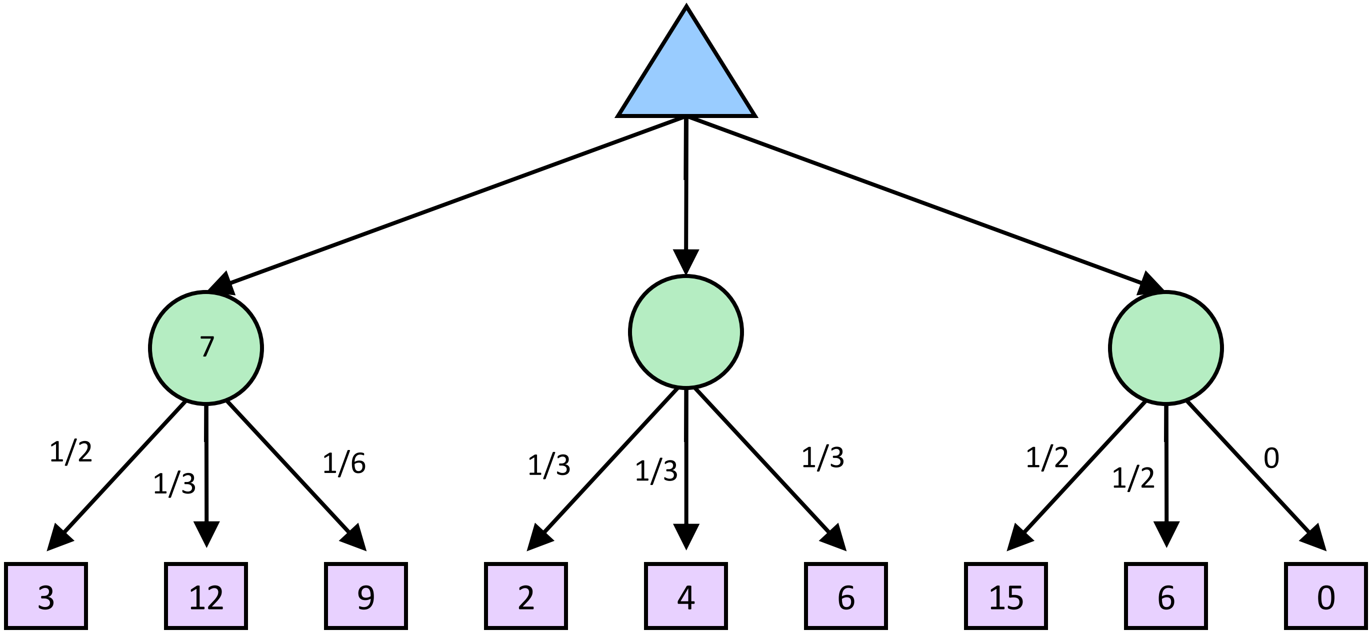
v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10

Expectimax Example

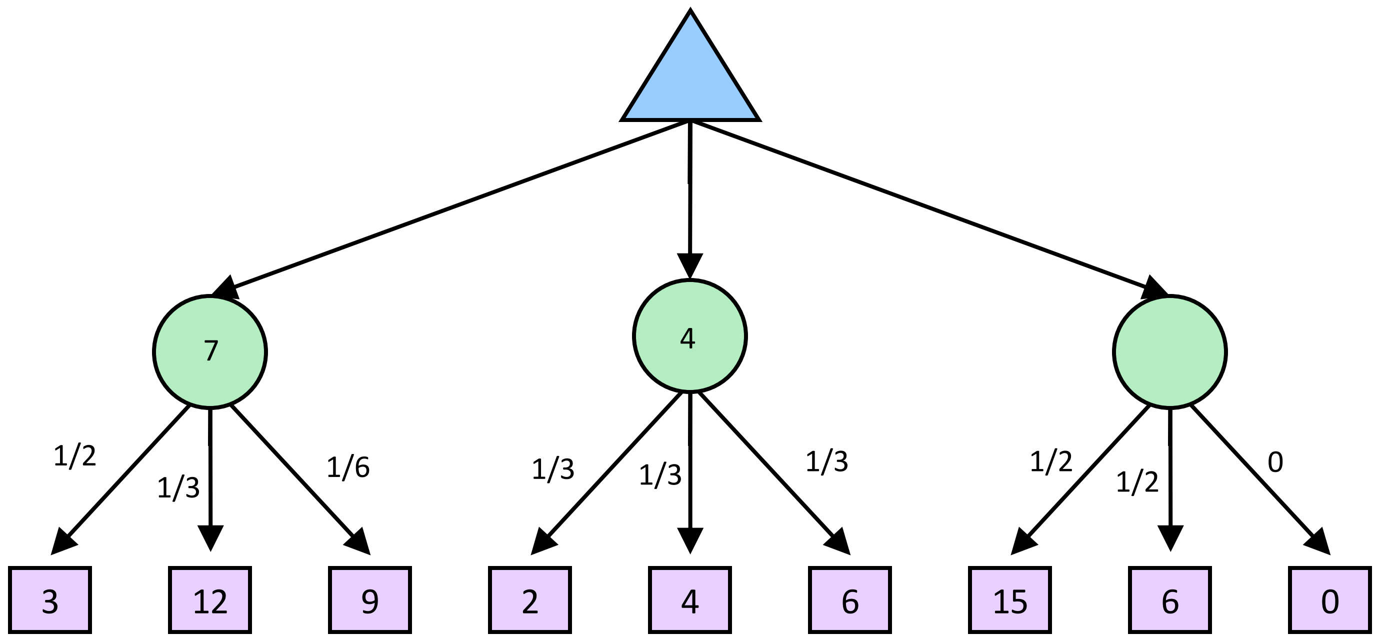


Example

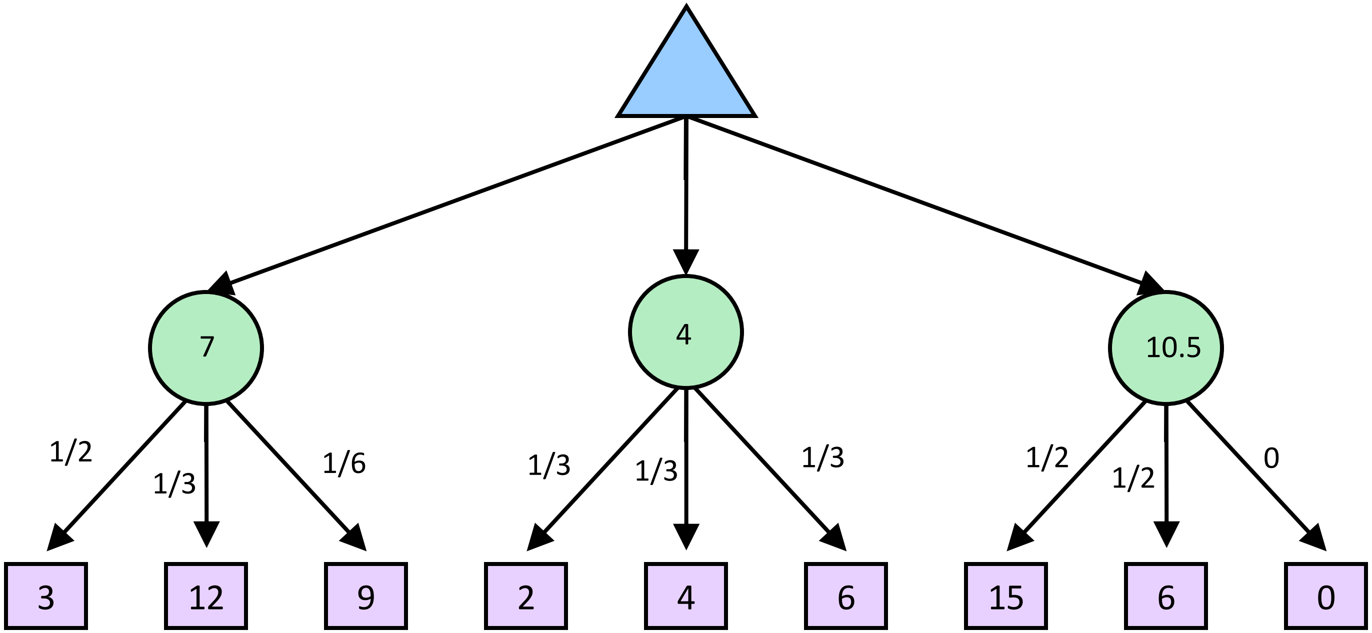
Expectimax Example



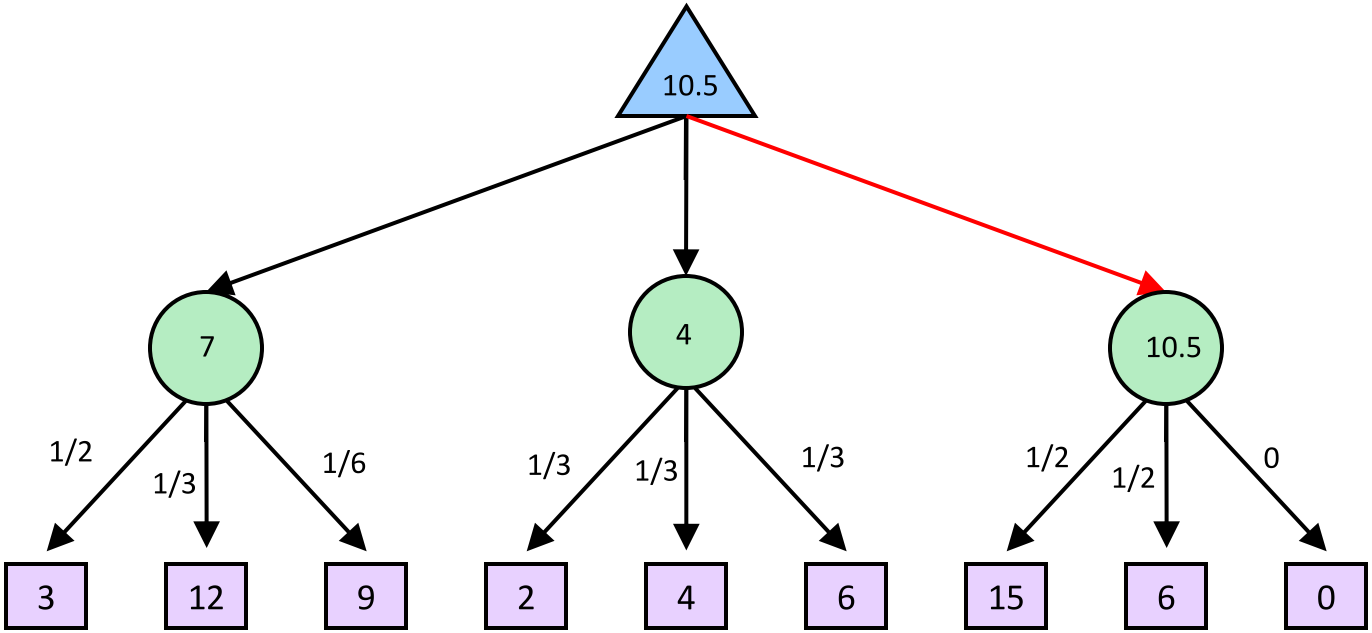
Expectimax Example



Expectimax Example



Expectimax Example



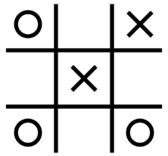
Model for chance nodes

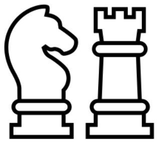
Model for chance nodes

- Expectiminimax search assumes to have a probabilistic model of how the opponent (or environment) will behave in any state
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a chance node for any outcome out of our control: opponent or environment

Types of Games

- Classified over different axes:
  - Number of players
  - Zero sum
  - Deterministic or stochastic
  - Perfect or imperfect information





Games with imperfect information

- Games could be partially observable
  - Result into belief states
- Solution for each state in the belief as deterministic game (with all techniques we have seen)
  - Instead of solving all of them, randomly samples states in the belief to solve them

Commonsense - Information is important

Mondays:

- Road A leads to 1 gold. Road B leads to a fork.
- Go left from the fork, 100 gold. Go right, die.
- *Optimal strategy: B*

Tuesdays:

- Road A leads to 1 gold. Road B leads to a fork.
- Go left from the fork, die. Go right, 100 gold.
- *Optimal strategy: B*

I can't remember what day it is. *Choose B?*

Commonsense - Information is important

Mondays:

- Road A leads to 1 gold. Road B leads to a fork.
- Go left from the fork, 100 gold. Go right, die.
- *Optimal strategy: B*

Tuesdays:

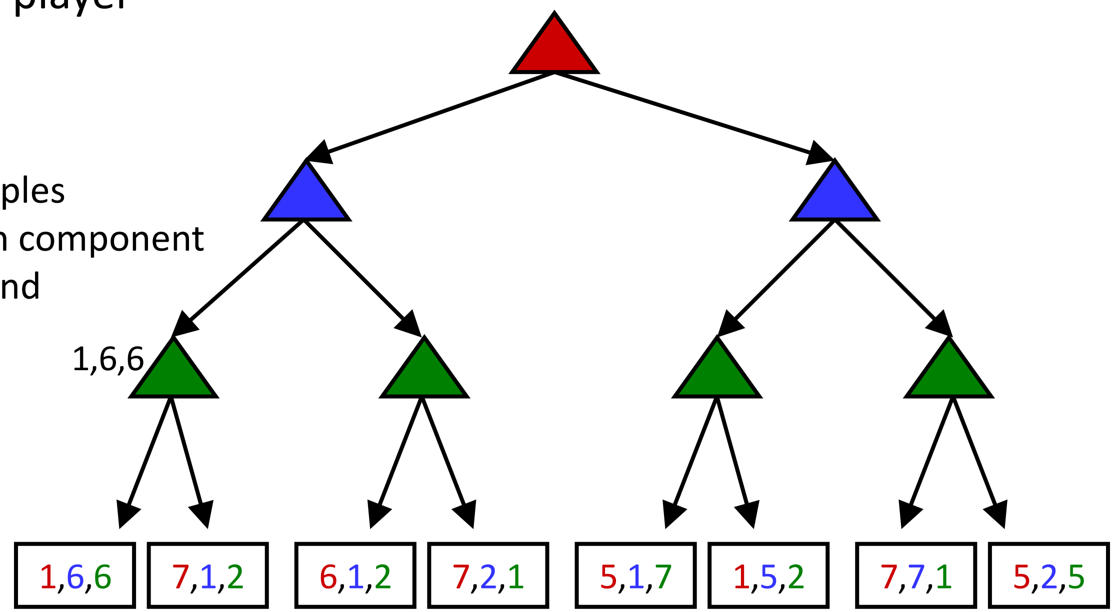
- Road A leads to 1 gold. Road B leads to a fork.
- Go left from the fork, die. Go right, 100 gold.
- *Optimal strategy: B*

I can't remember what day it is. *Choose B?*

With partial observability, value of an action depends on belief state of the agent -> agent might try to reduce that uncertainty

Multi-Agent Utilities

- With multiple players, the tree can be extended with additional nodes corresponding to each player
- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically



Summary

- Stochastic games introduce chance nodes
  - Expectiminimax for finding the solution that maximizes the average case
  - A probabilistic model is needed
- Partially observable games result in belief states

Next

- Non-zero sum games