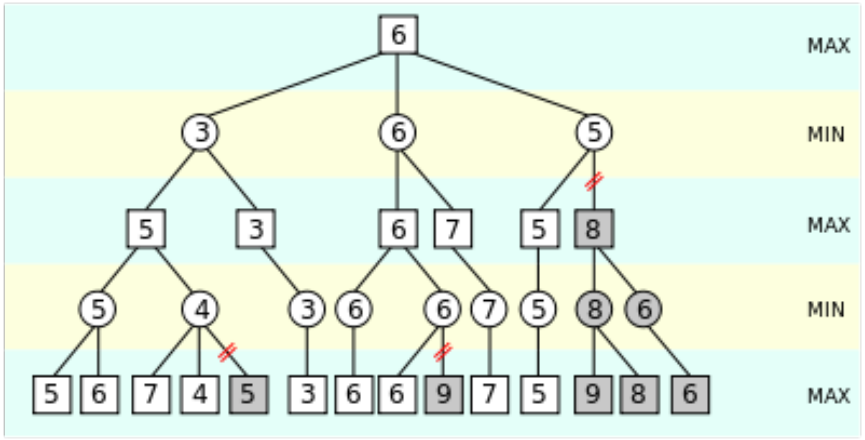
CS440/ECE448 Lecture 19: Alpha-Beta and Expectiminimax

Mark Hasegawa-Johnson, 4/2021

Including slides by Svetlana Lazebnik, 2/2017



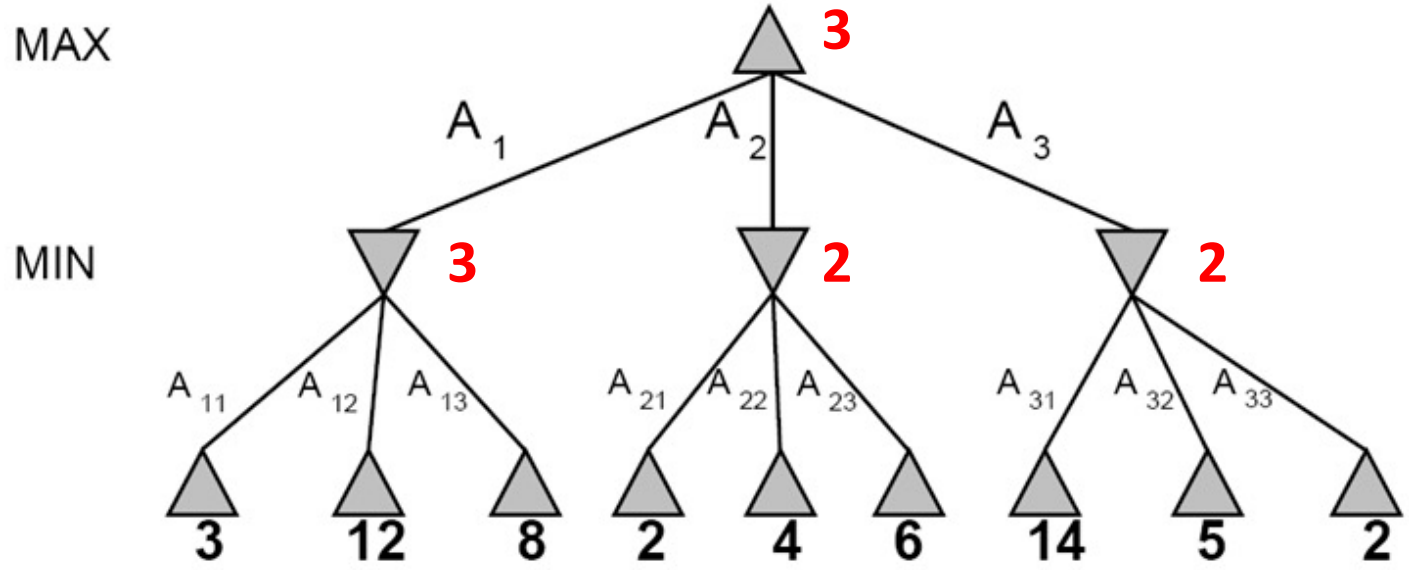
Alpha-beta pruning.
Jex9999, GFDL, 2007
https://commons.wikimedia.org/wiki/File:AB_pruning.svg

A contemporary backgammon set. Public domain photo by Manuel Hegner, 2013,
https://commons.wikimedia.org/w/index.php?curid=25006945

Outline

- Alpha-beta pruning
  - alpha (α) is the highest score that MAX knows how to force MIN to accept
  - beta (β) is the lowest score the MIN knows how to force MAX to accept
  - With optimum move ordering, computational complexity is $O\{n^{d/2}\}$
- Expectiminimax: Minimax search for games of chance
  - Besides MIN and MAX, there's one more player: CHANCE
  - The value of a CHANCE node is the expected value of its daughters
  - Number of levels is doubled, branching factor is large

Minimax Search



- **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

Computational complexity of minimax

- Suppose that, at each game state, there are n possible moves
- Suppose we search to a depth of d
- Then the computational complexity is $O\{n^d\}$!

Basic idea of alpha-beta pruning

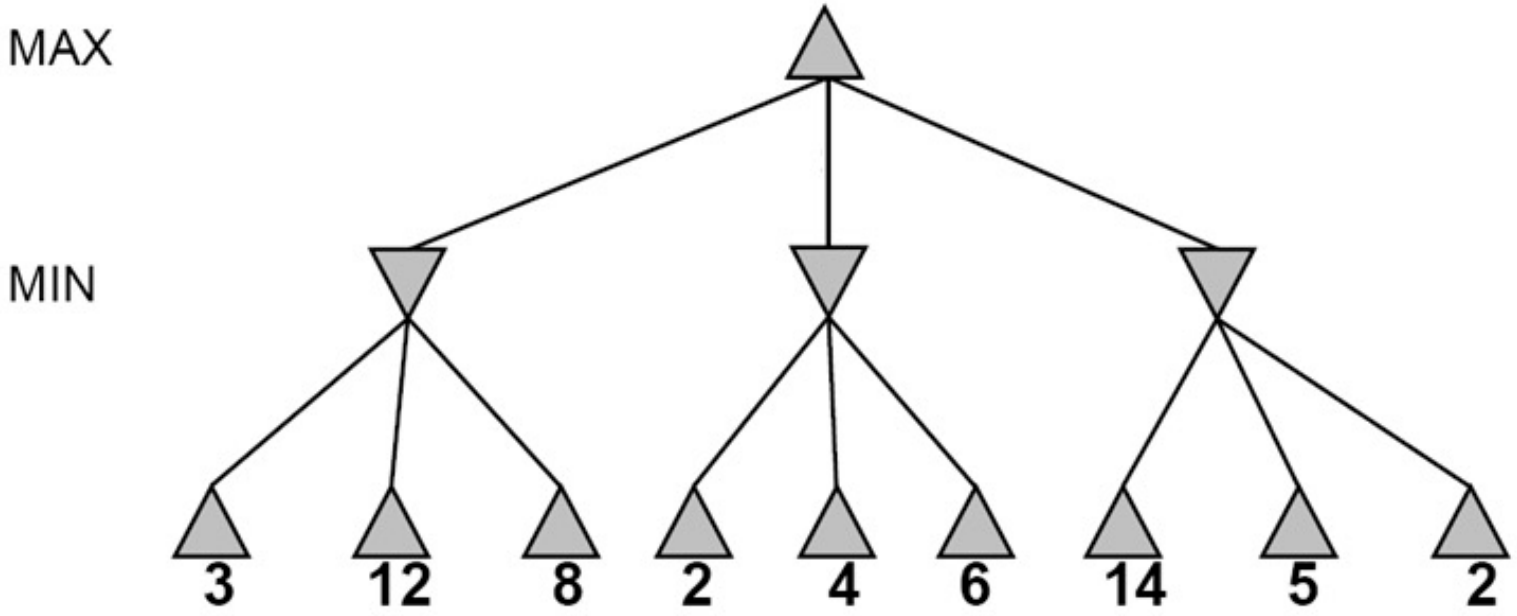
- Computational complexity of minimax is $O\{n^d\}$
- There is no known algorithm to make it polynomial time
- But… can we reduce the exponent? For example, could we make the complexity $O\{n^{d/2}\}$?
- If we could do that, then it would become possible to search twice as far, using the same amount of computation. This could be the difference between a beginner chess player vs. a grand master.

Basic idea of alpha-beta pruning

- The basic idea of alpha-beta pruning is to reduce the complexity of minimax from $O\{n^d\}$ to $O\{n^{d/2}\}$.
- We can do this by only evaluating half of the levels.
- How can we ”only evaluate half the levels” without any reduction of accuracy?

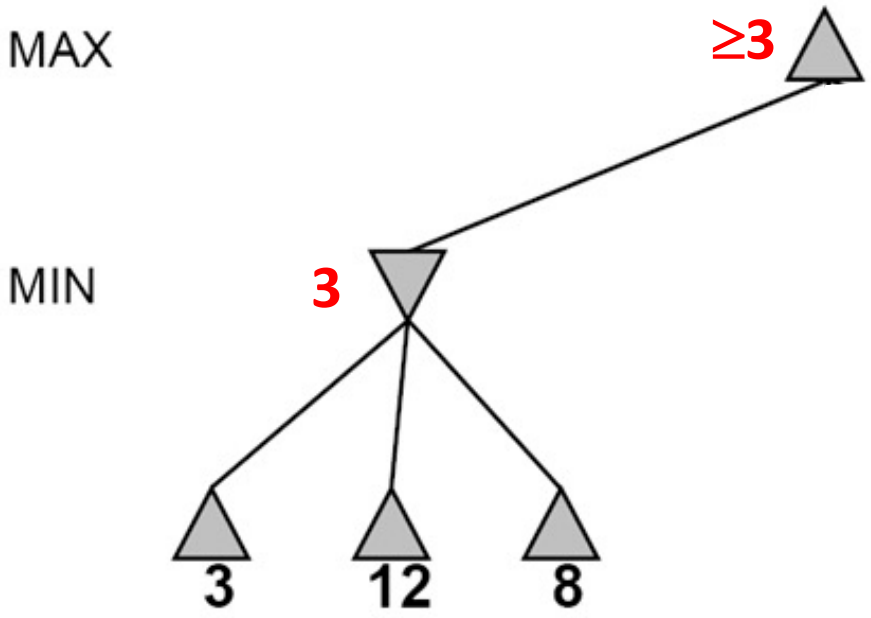
Alpha-beta pruning

Why it works: It is possible to compute the exact minimax decision without expanding every node in the game tree



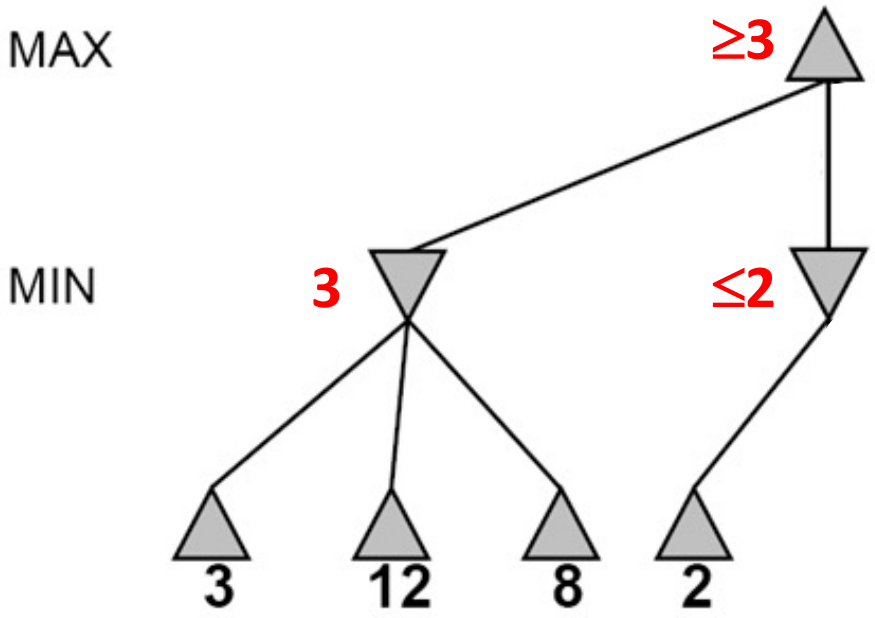
Alpha-beta pruning

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



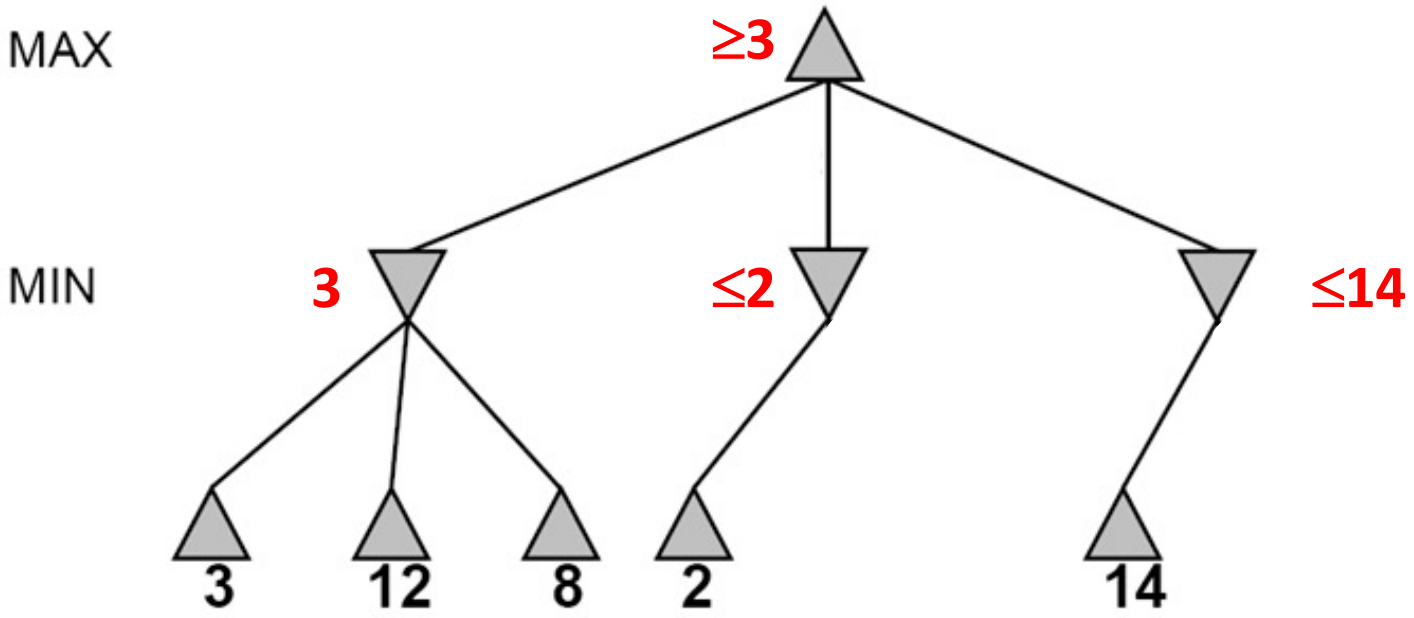
Alpha-beta pruning

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



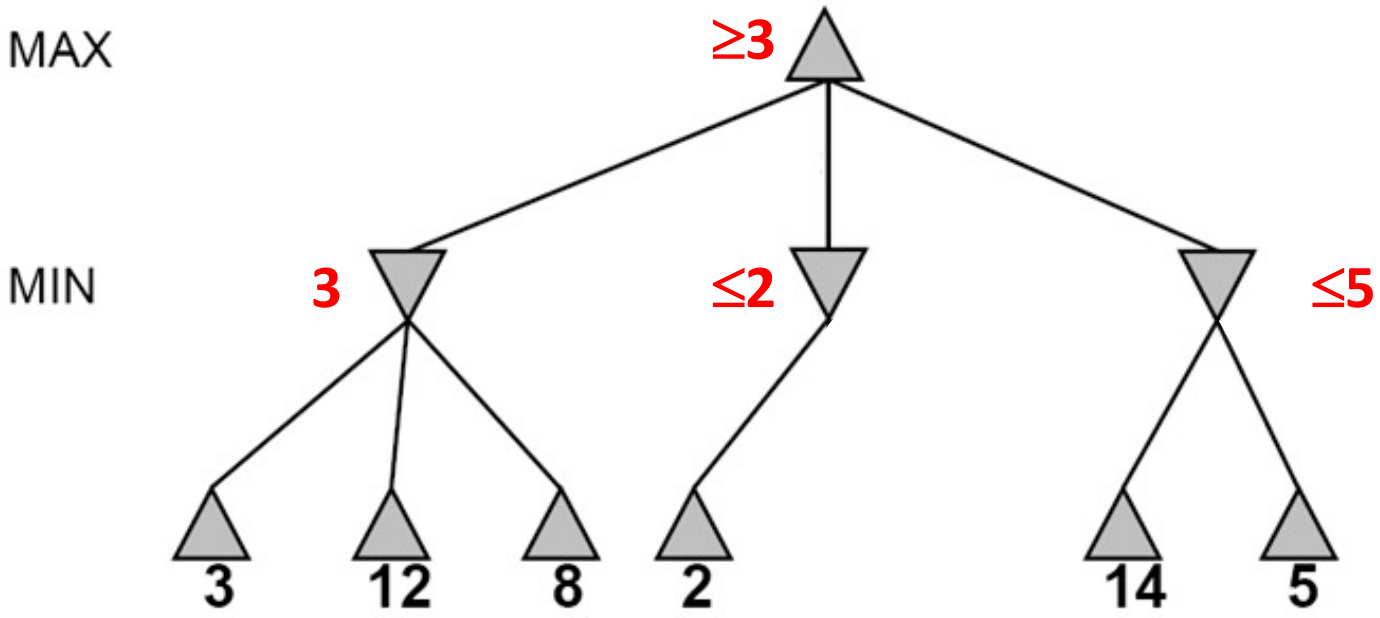
Alpha-beta pruning

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



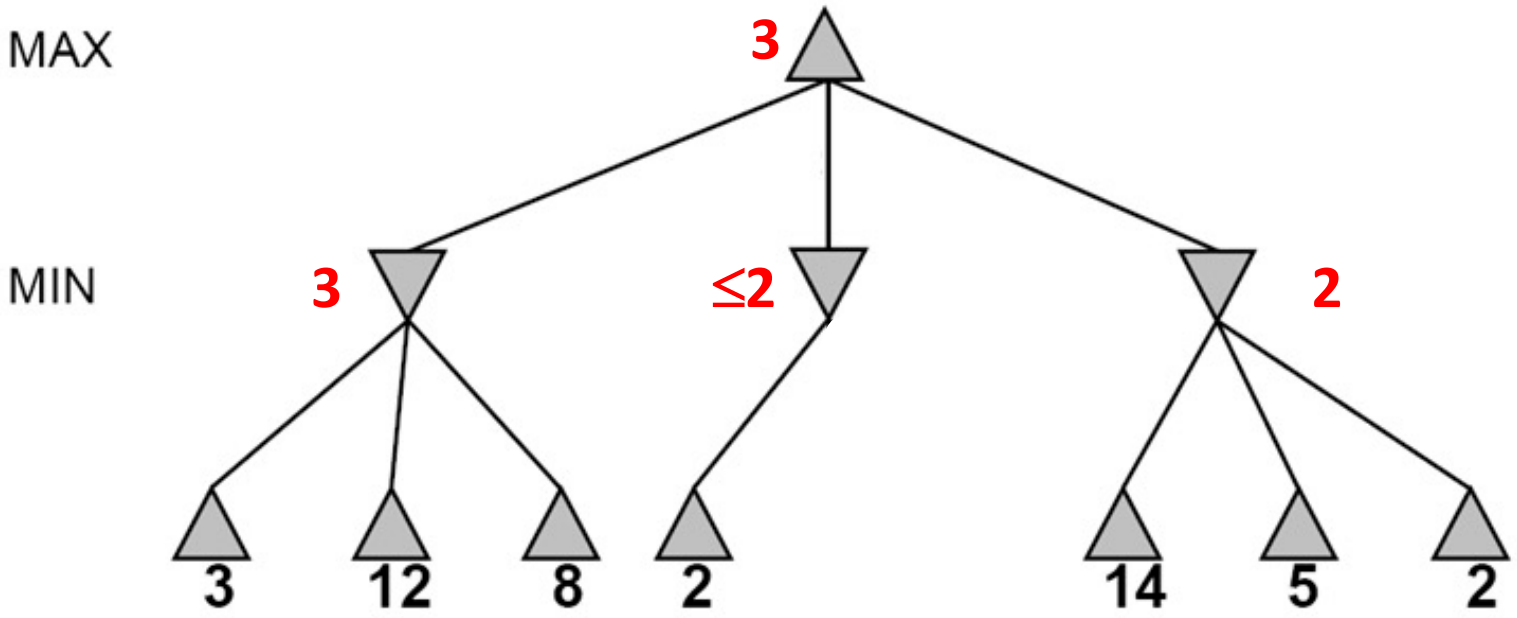
Alpha-beta pruning

- 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



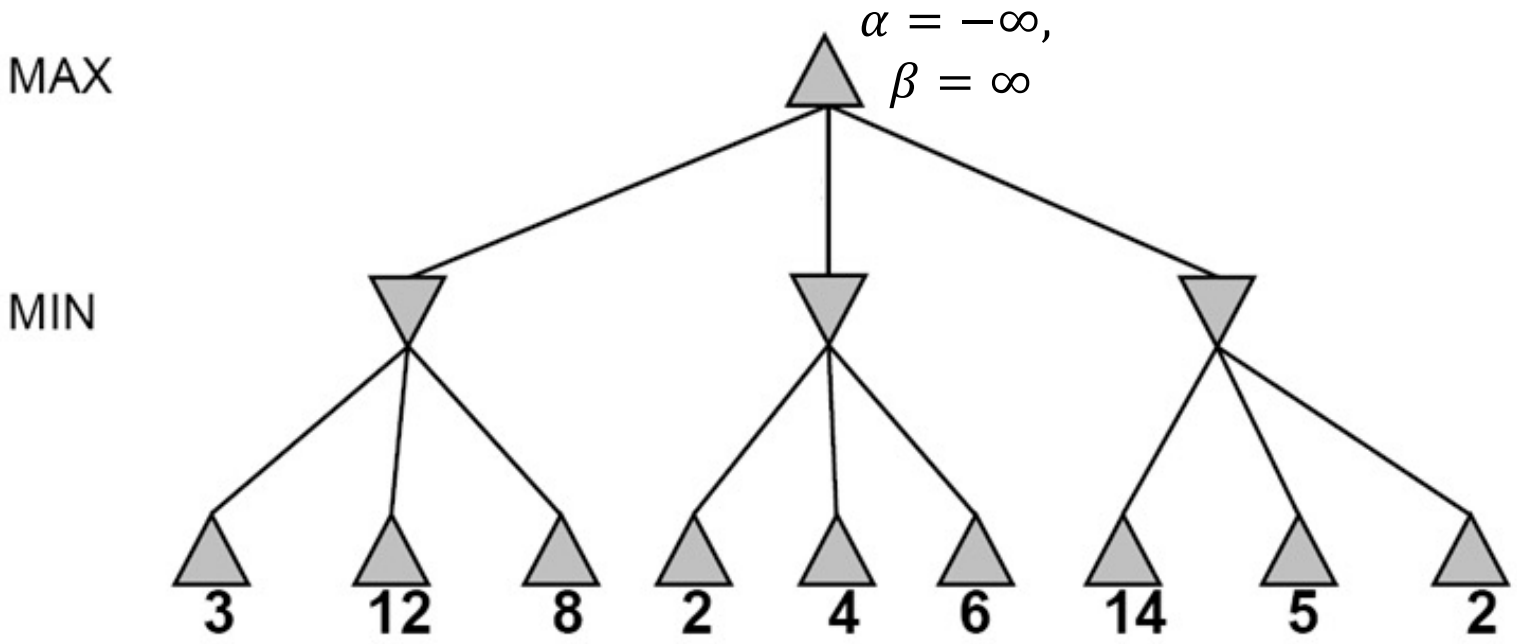
The pruning thresholds, alpha and beta

Alpha-beta pruning requires us to keep track of two pruning thresholds, alpha and beta.

- alpha (α) is the highest score that MAX knows how to force MIN to accept.
- beta (β) is the lowest score that MIN knows how to force MAX to accept.
- $\alpha < \beta$

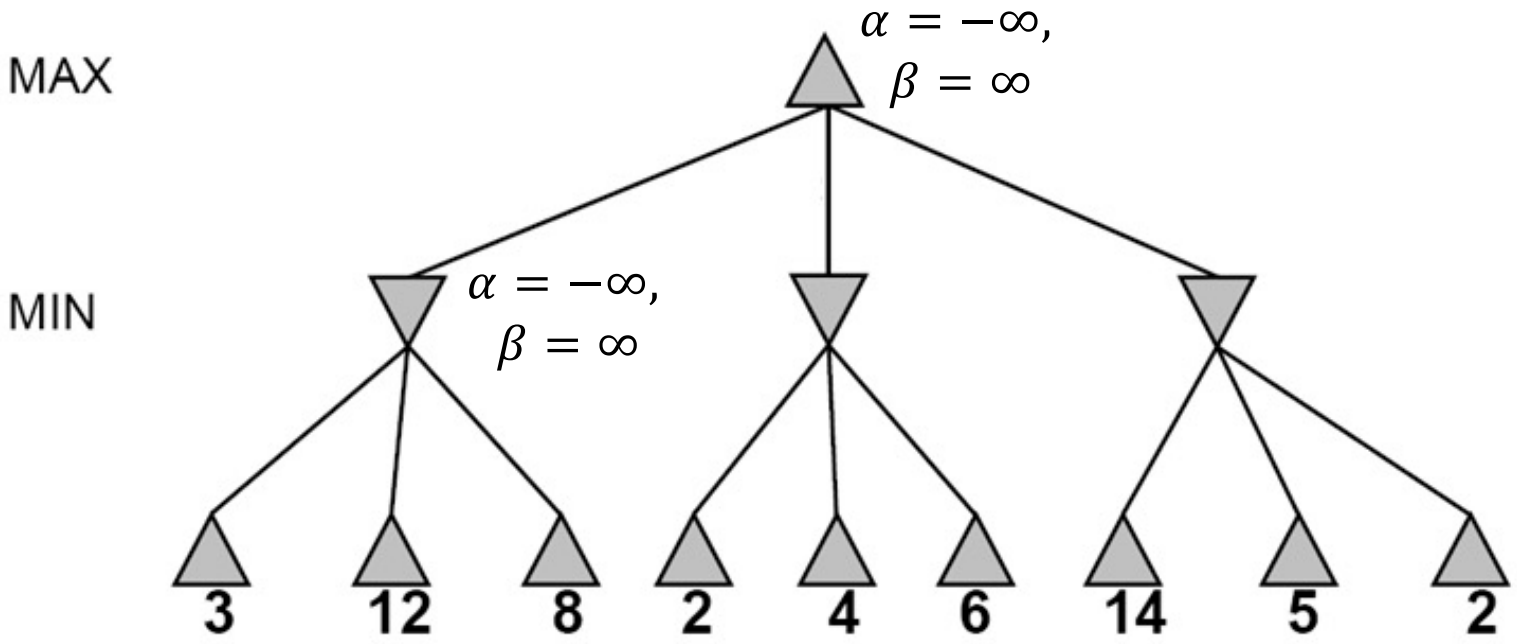
Alpha-beta pruning

Why it works: It is possible to compute the exact minimax decision without expanding every node in the game tree



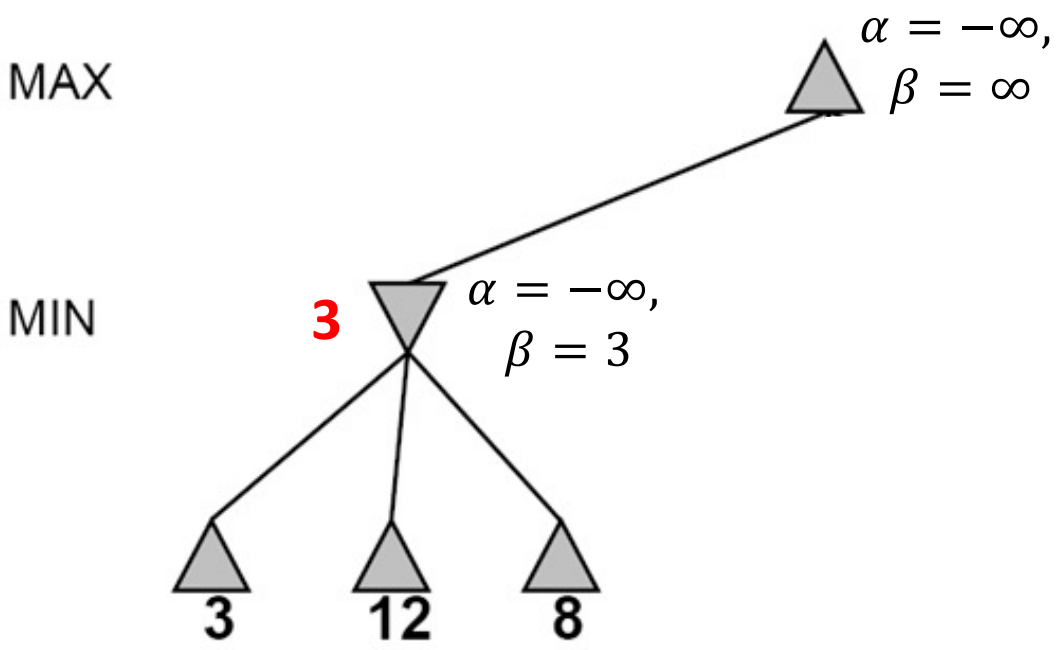
Alpha-beta pruning

Why it works: It is possible to compute the exact minimax decision without expanding every node in the game tree



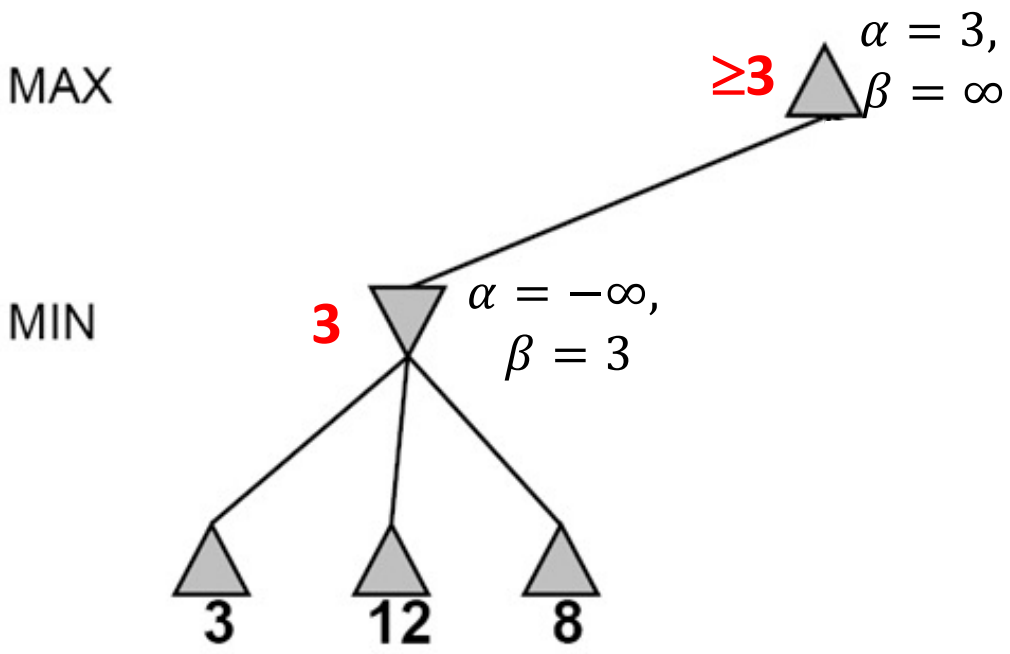
Alpha-beta pruning

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



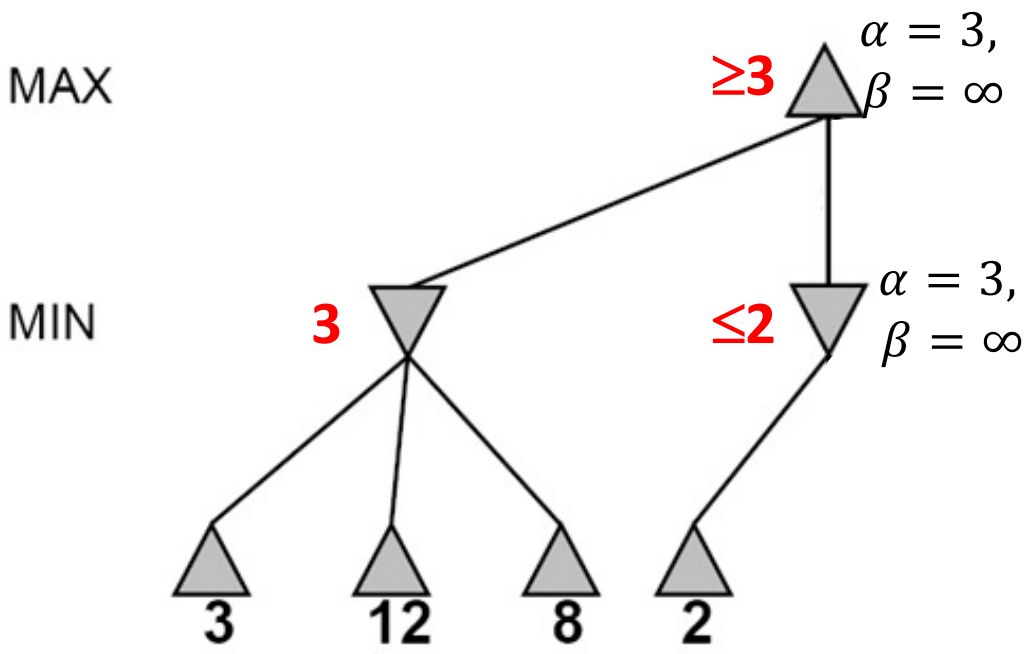
Alpha-beta pruning

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



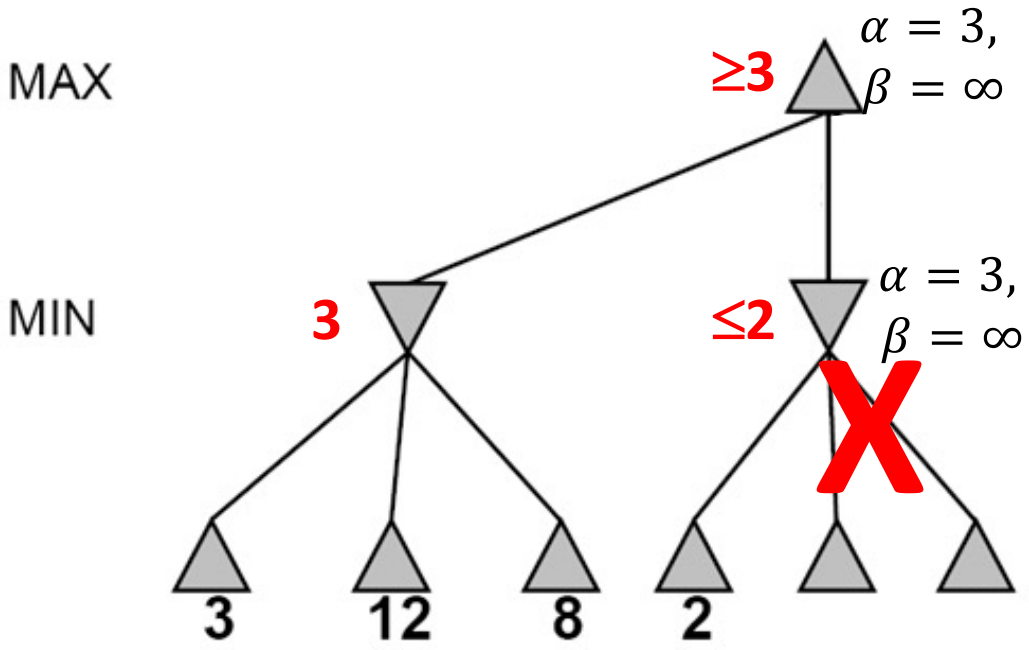
Alpha-beta pruning

- 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

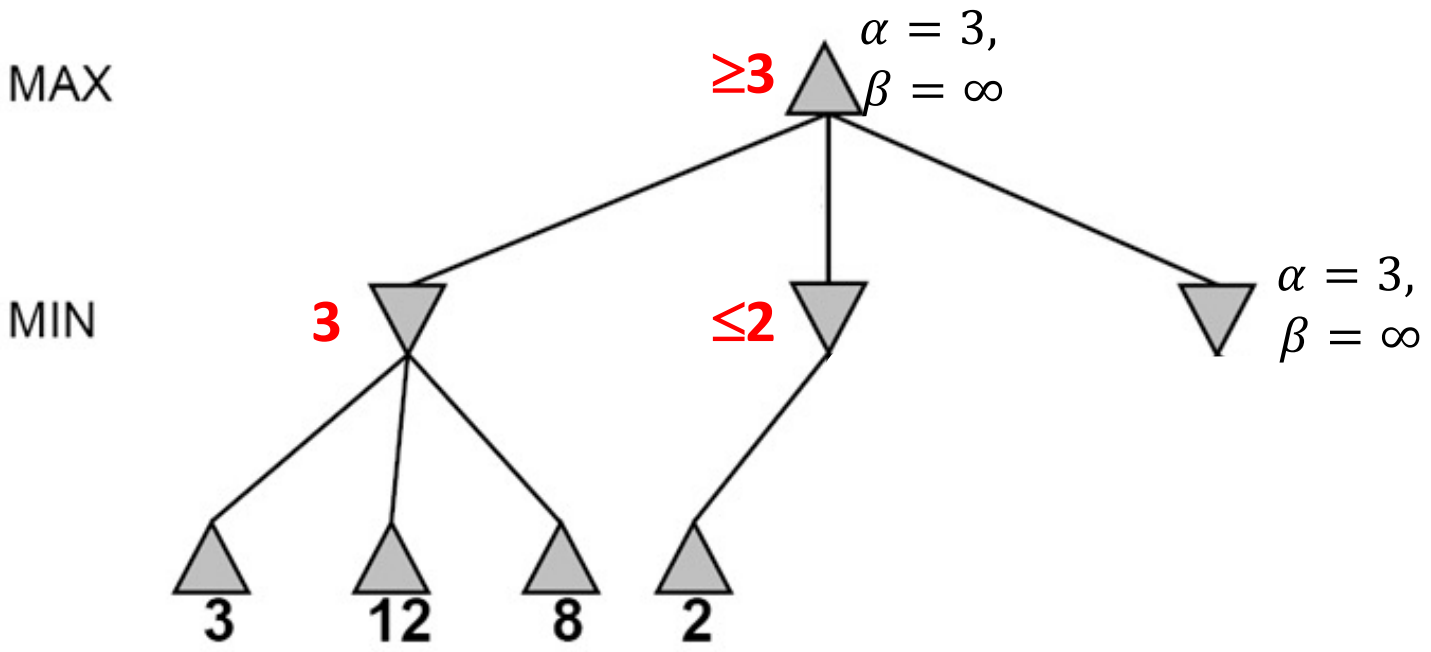


PRUNE!

- If MAX lets us get to this state, then MIN would achieve a final score <=2
- Therefore MAX will never let us get to this state!
- Therefore there's no need to score the remaning children of this node.

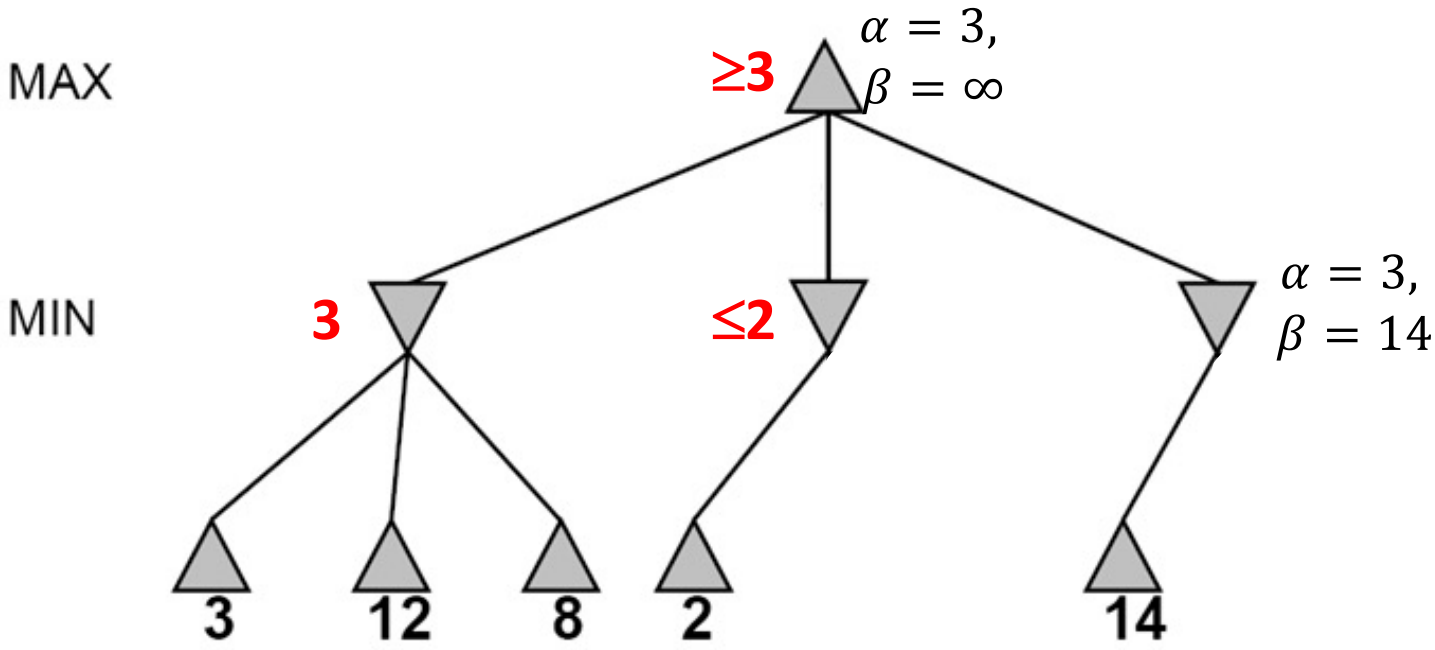
Alpha-beta pruning

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



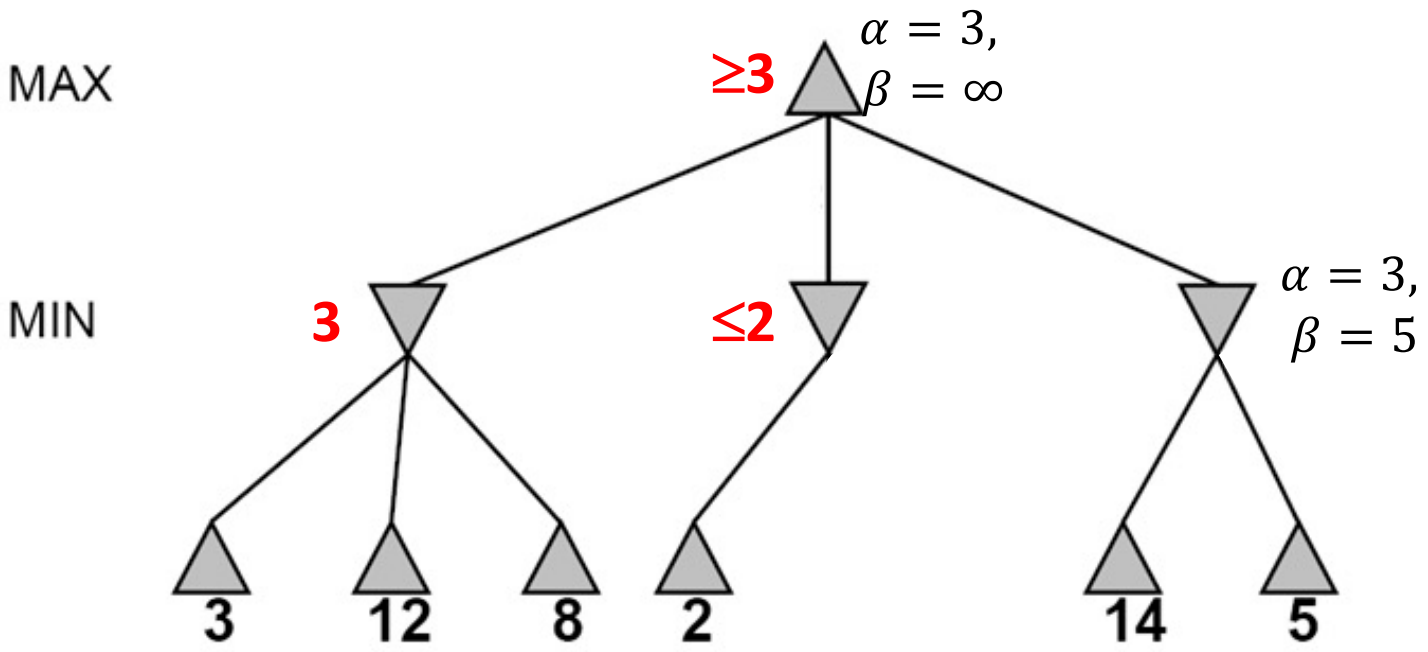
Alpha-beta pruning

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



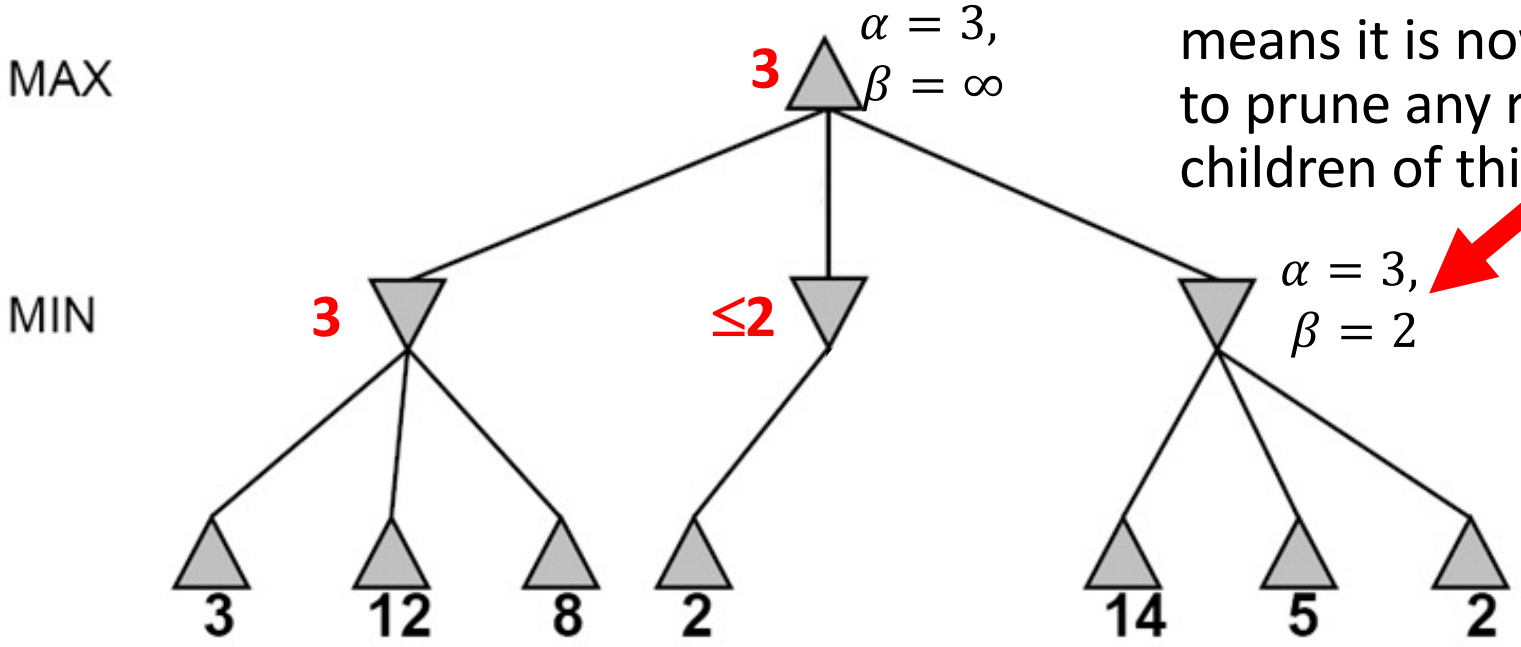
Alpha-beta pruning

- 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



Violation of $\alpha < \beta$ means it is now possible to prune any remaining children of this node.

Outline

- Alpha-beta pruning
  - alpha (α) is the highest score that MAX knows how to force MIN to accept
  - beta (β) is the lowest score the MIN knows how to force MAX to accept
  - With optimum move ordering, computational complexity is $O\{n^{d/2}\}$
- Expectiminimax: Minimax search for games of chance
  - Besides MIN and MAX, there's one more player: CHANCE
  - The value of a CHANCE node is the expected value of its daughters
  - Number of levels is doubled, branching factor is large

Computational complexity of alpha-beta pruning

- The basic idea of alpha-beta pruning is to reduce the complexity of minimax from $O\{n^d\}$ to $O\{n^{d/2}\}$.
- That can be done with no loss of accuracy,
- … but only if the children of any given node are optimally ordered.

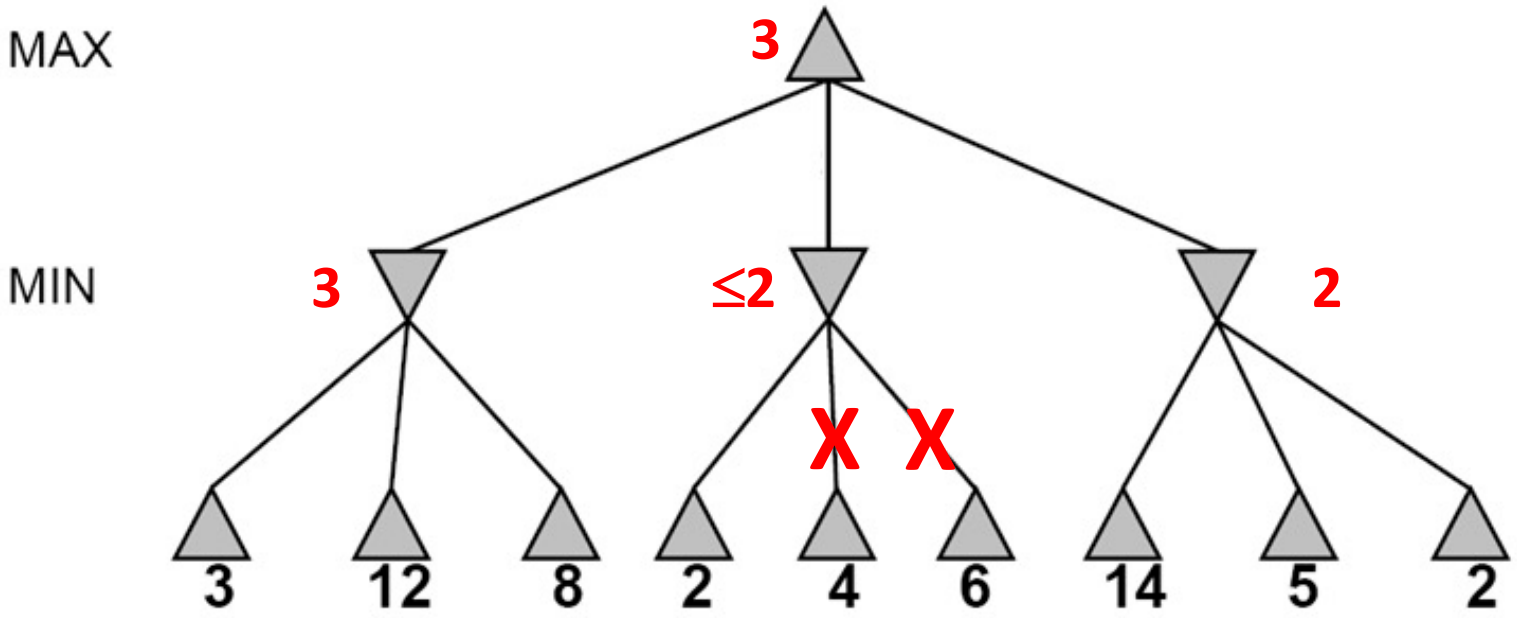
Optimal ordering

Minimum computational complexity ($O\{n^{d/2}\}$) is only achieved if:

- The children of a MAX node are evaluated, in order, starting with the highest-value child.
- The children of a MIN node are evaluated, in order, starting with the lowest-value child.

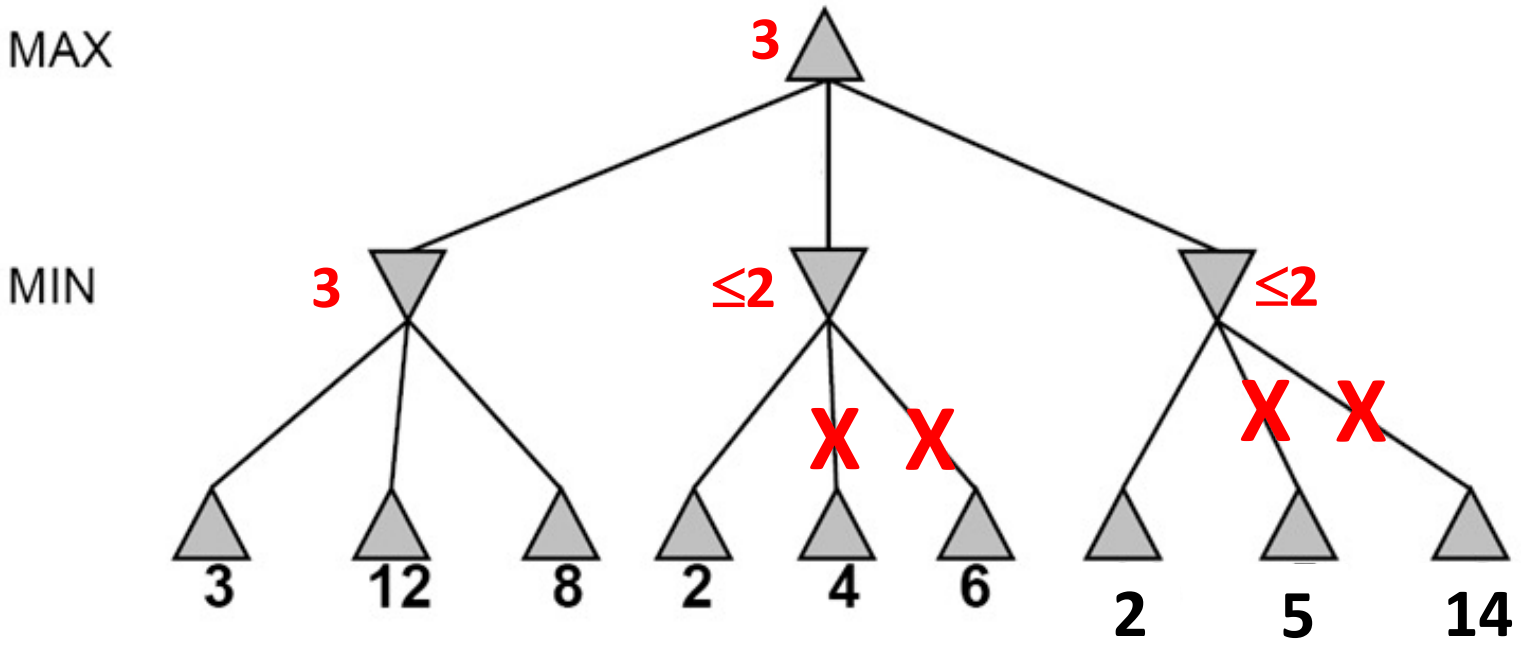
Non-optimal ordering

In this tree, the moves are not optimally ordered, so we were only able to prune two nodes.



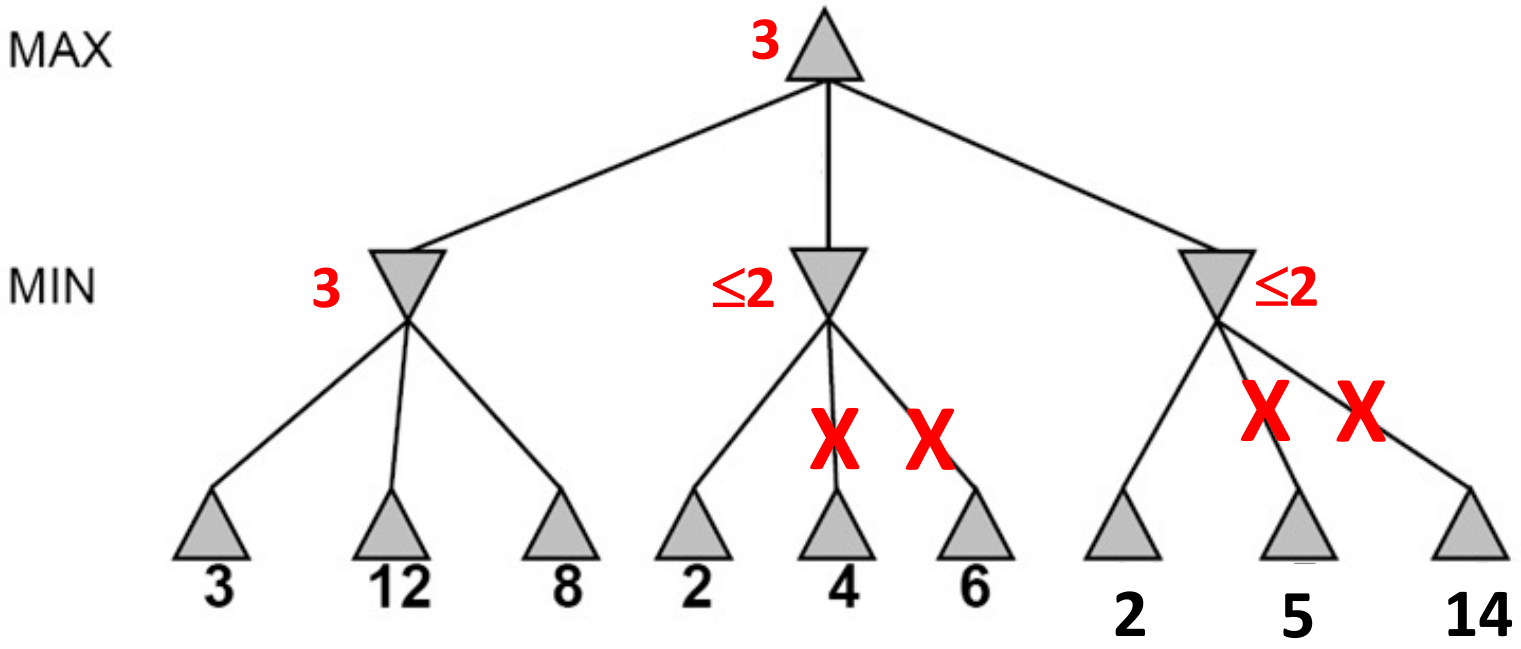
Optimal ordering

In this tree, the moves ARE optimally ordered, so we are able to prune four nodes (out of nine).

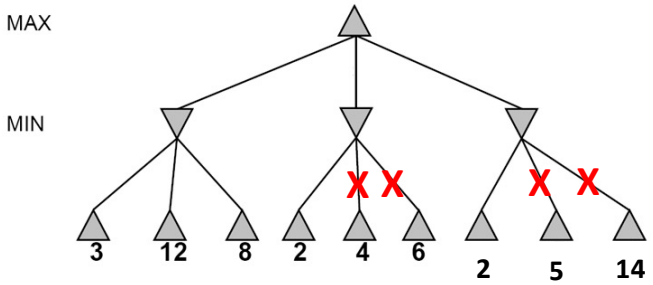


Optimal ordering

In this tree, the moves ARE optimally ordered, so we are able to prune four nodes (out of nine).



Computational Complexity

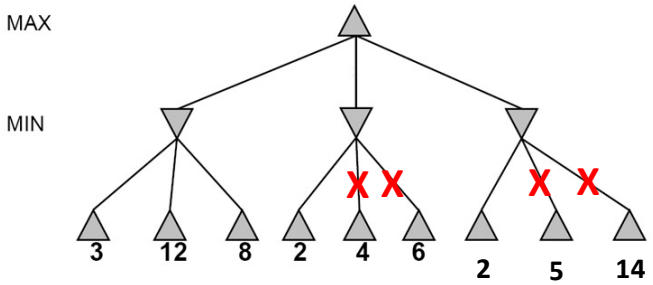


Consider a sequence of two levels, with n moves per level, and with optimal ordering.

- There are n^2 terminal nodes.
- Alpha-beta will evaluate all the daughters of the first daughter: n nodes.
- Alpha-beta will also evaluate the first daughter of each non-first daughter: $n-1$ nodes.
- In total, alpha-beta will evaluate $2n-1$ out of every n^2 nodes.
- For a tree of depth d, the number of nodes evaluated by alpha-beta is

$$(2n-1)^{d/2} = O\{n^{d/2}\}$$

Computational Complexity



…but wait… this means we need to know, IN ADVANCE, which move has the highest value, and which move has the lowest value!!

- Obviously, it is not possible to know the true value of a move without evaluating it.
- However, heuristics often are pretty good.
- We use the heuristic to decide which move to evaluate first.
- For games like chess, with good heuristics, complexity of alpha-beta is closer to $O\{n^{d/2}\}$ than to $O\{n^{d}\}$.

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

Outline

- Alpha-beta pruning
  - alpha (α) is the highest score that MAX knows how to force MIN to accept
  - beta (β) is the lowest score the MIN knows how to force MAX to accept
  - With optimum move ordering, computational complexity is $O\{n^{d/2}\}$
- Expectiminimax: Minimax search for games of chance
  - Besides MIN and MAX, there's one more player: CHANCE
  - The value of a CHANCE node is the expected value of its daughters
  - Number of levels is doubled, branching factor is large

Stochastic games

How can we incorporate dice throwing into the game tree?

Minimax

State evolves deterministically (when a player acts, that action uniquely determines the following state).

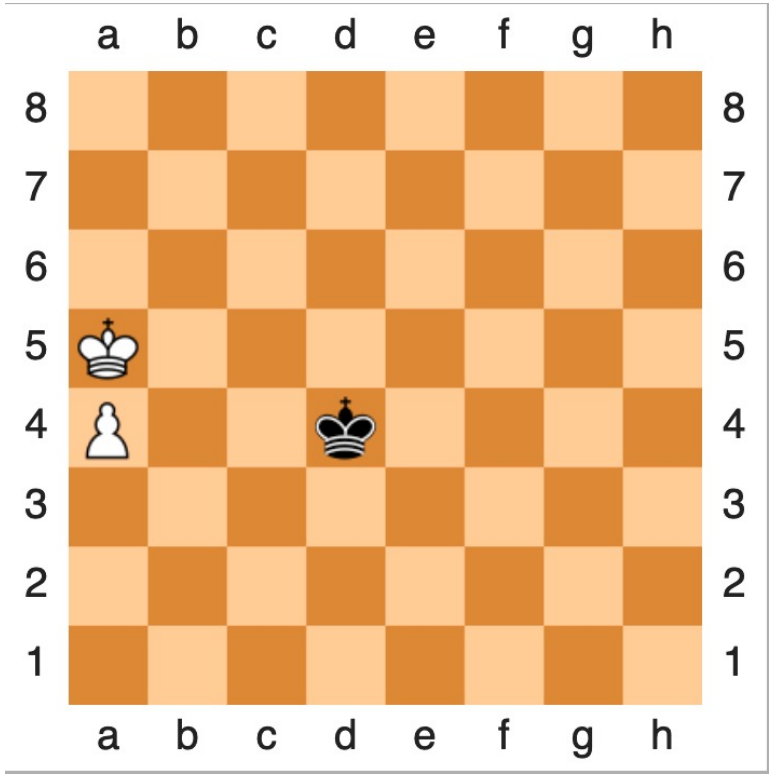
Current state is visible to both players.

Each player tries to maximize his or her own reward:

- **Maximize** (over all possible moves I can make) the
- **Minimum** (over all possible moves Min can make) of the resulting utility:

$$U(s) = \max_{s' \in C(s)} U(s')$$

$$U(s') = \min_{s'' \in C(s')} U(s'')$$



Expectiminimax

State evolves **stochastically** (when a player acts, the game changes RANDOMLY, with a probability distribution $P(s'|s,a)$ that depends on the action, a).

Current state, s, is visible to the player.

The player tries to maximize his or her own reward:

- **Maximize** (over all possible moves I can make) the
- **Expected value** (over all possible successor states) of the resulting utility:

$$Q(s,a) = \sum_{s'} P(s'|s,a)U(s')$$

Expectiminimax

State evolves **stochastically** (when a player acts, that action influences the state transition probability).

Current state is visible to both players.

Each player tries to maximize his or her own reward:

- **Maximize** (over all possible moves I can make) the
- **Minimum** (over all possible moves Min can make) of the
- **Expected value** (over all possible successor states) of the resulting utility:

$$U(s) = \max_{a} \sum_{s'} P(s'|s,a)U(s')$$

$$U(s') = \min_{a'} \sum_{s''} P(s''|s',a')U(s'')$$

Expectiminimax: notation

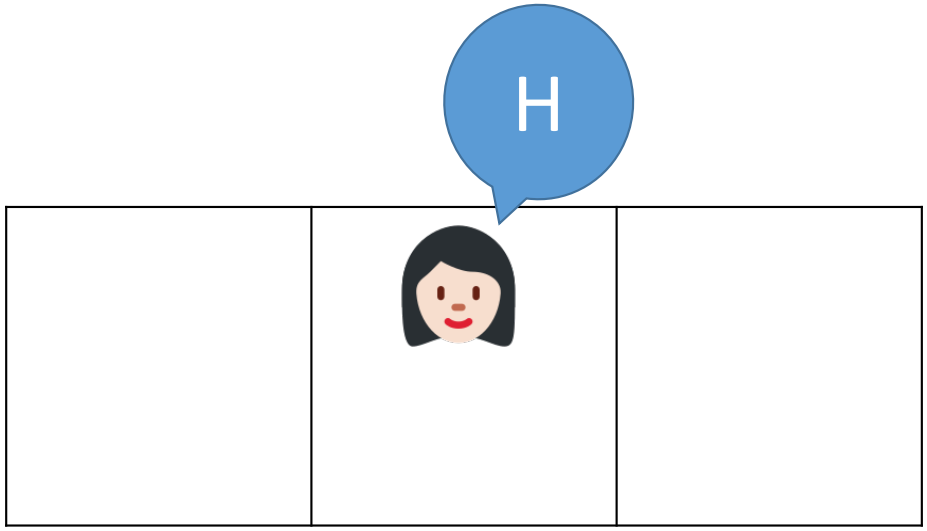
= MAX node. $U(s) = \max_{a \in A(s)} Q(s,a)$

= MIN node. $U(s) = \min_{a \in A(s)} Q(s,a)$

= Chance node. $Q(s,a) = \sum_{s'} P(s'|s,a)U(s')$

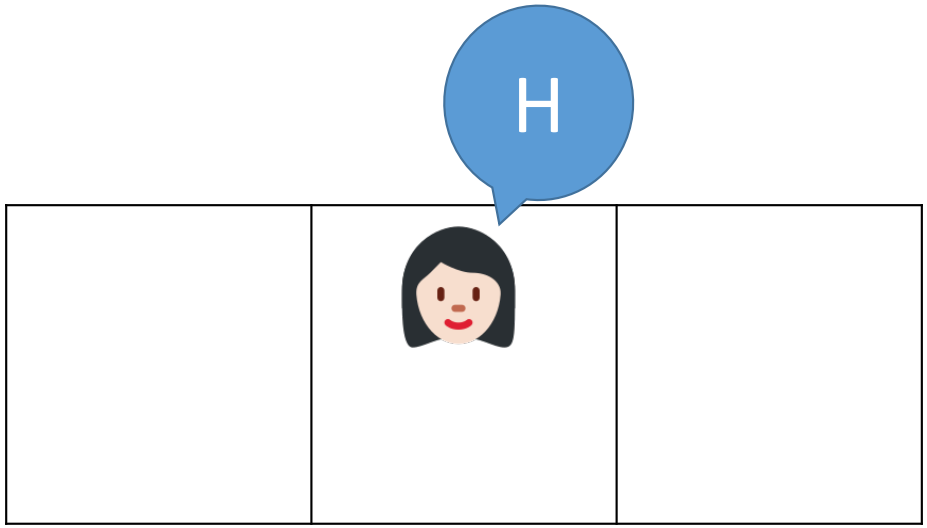
Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.



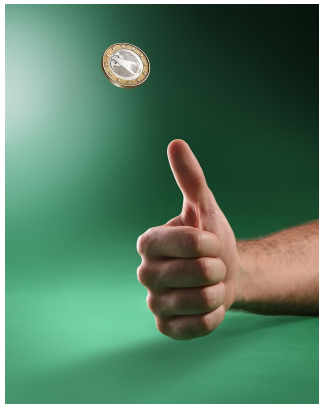
Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.

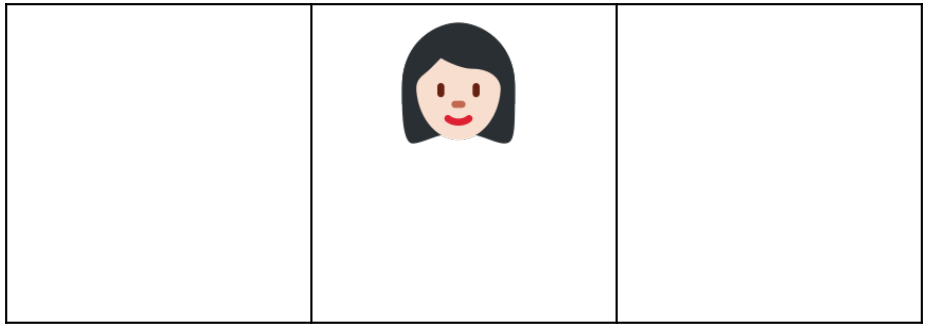


Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.



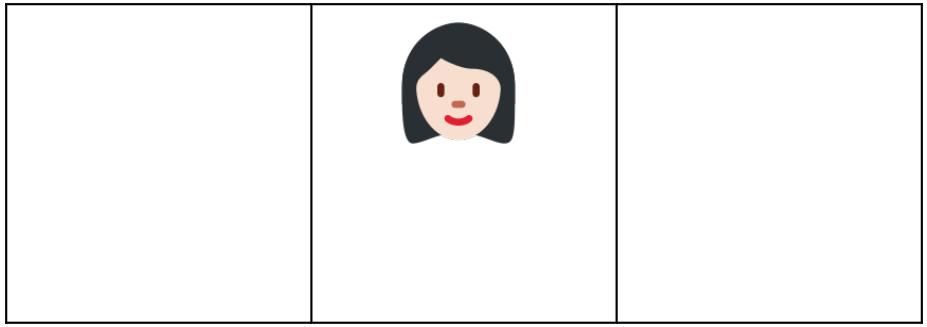
By ICMA Photos - Coin Toss, CC BY-SA 2.0, https://commons.wikimedia.org/w/index.php?curid=71147286



Emojis by Twitter, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=59974366

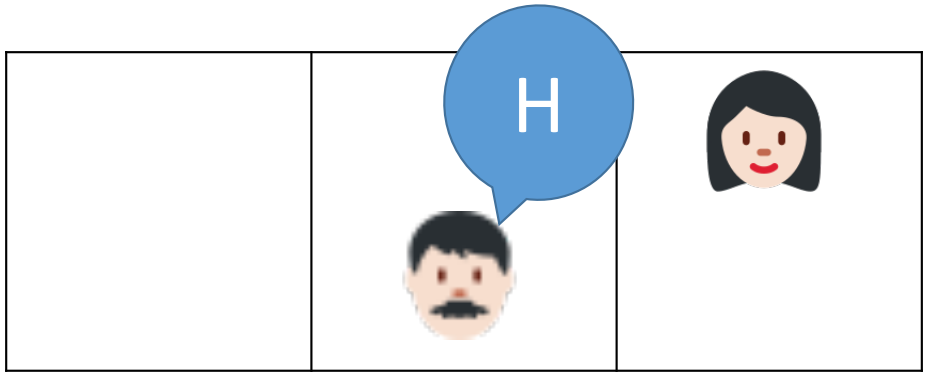
Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.



Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.
- MAX: Max decides whether to count heads (action H) or tails (action T) as a forward movement.



Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.
- MAX: Max decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: he flips a coin and moves his game piece in the direction indicated.

By NJR ZA - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4228918

By NJR ZA - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4228918

Emojis by Twitter, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=59974366

Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.
- MAX: Max decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: he flips a coin and moves his game piece in the direction indicated.

Reward: $2 to the winner, $0 for a draw.

By NJR ZA - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4228918

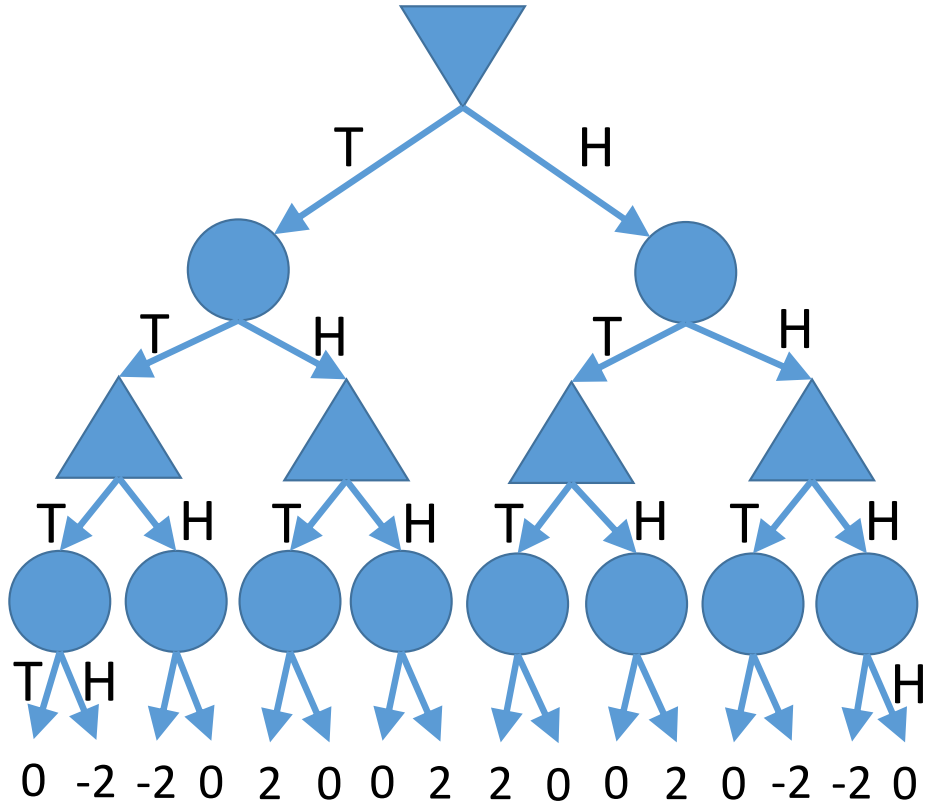
By NJR ZA - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4228918

Emojis by Twitter, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=59974366.
$2 By Bureau of Engraving and Printing: U.S. Department of the Treasury - own scanned, Public Domain, https://commons.wikimedia.org/w/index.php?curid=56299470

Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.
- MAX: Max decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: he flips a coin and moves his game piece in the direction indicated.

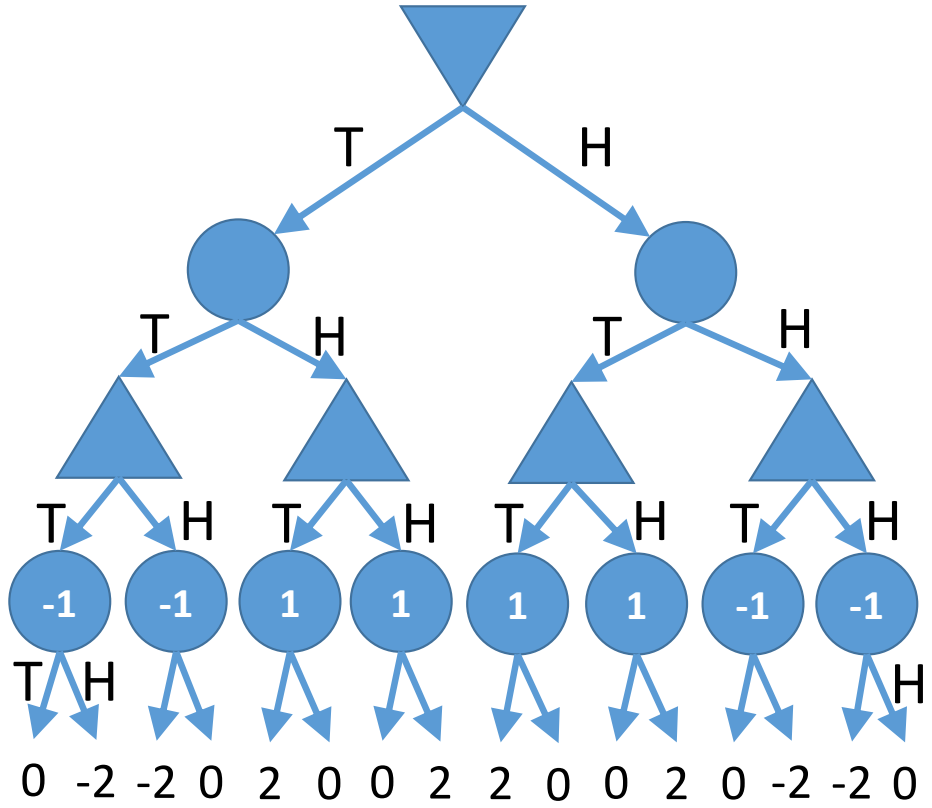
Reward: $2 to the winner, $0 for a draw.



Expectiminimax example

Chance node:

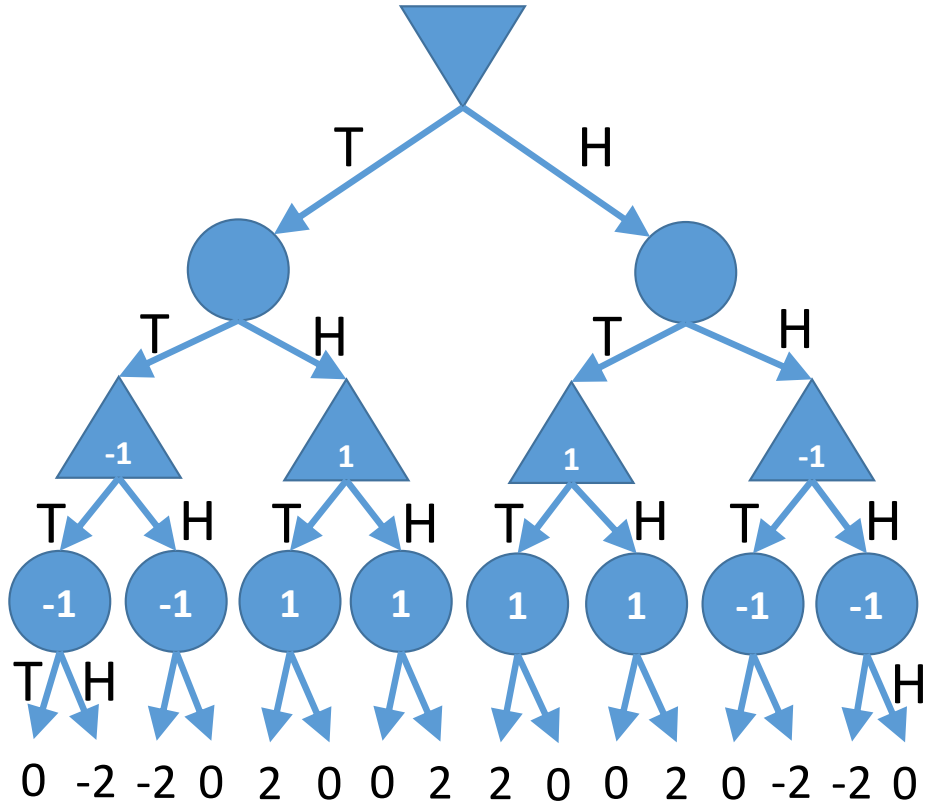
$$Q(s,a) = \sum_{s'} P(s'|s,a)U(s')$$



Expectiminimax example

Max node:

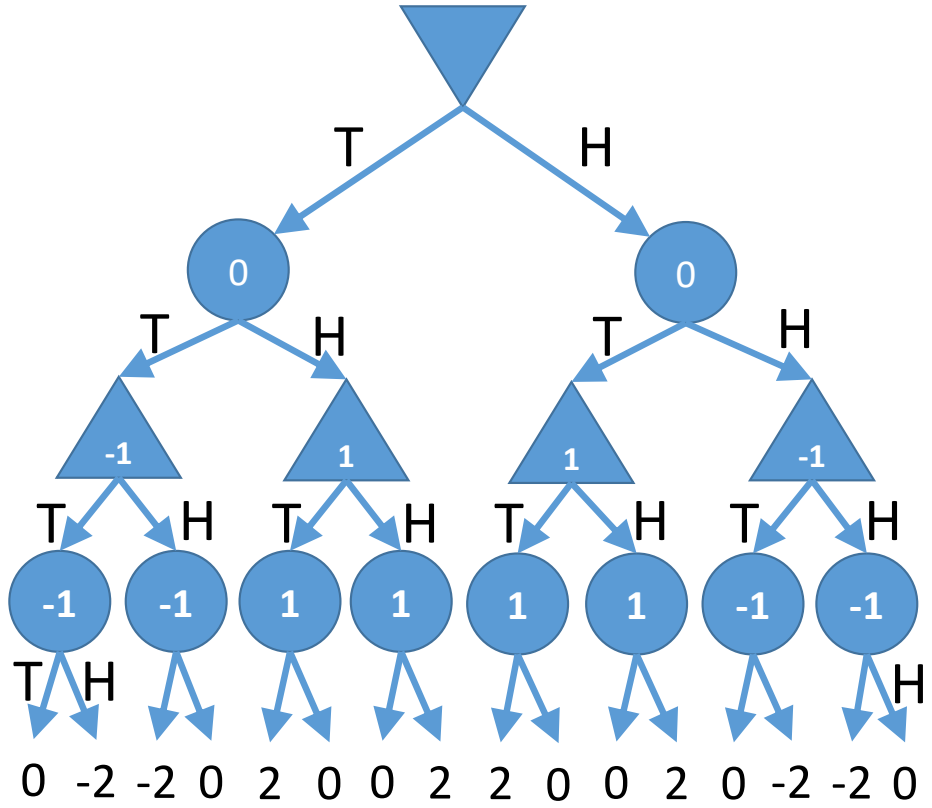
$$U(s) = \max_{a \in A(s)} Q(s, a)$$



Expectiminimax example

Chance node:

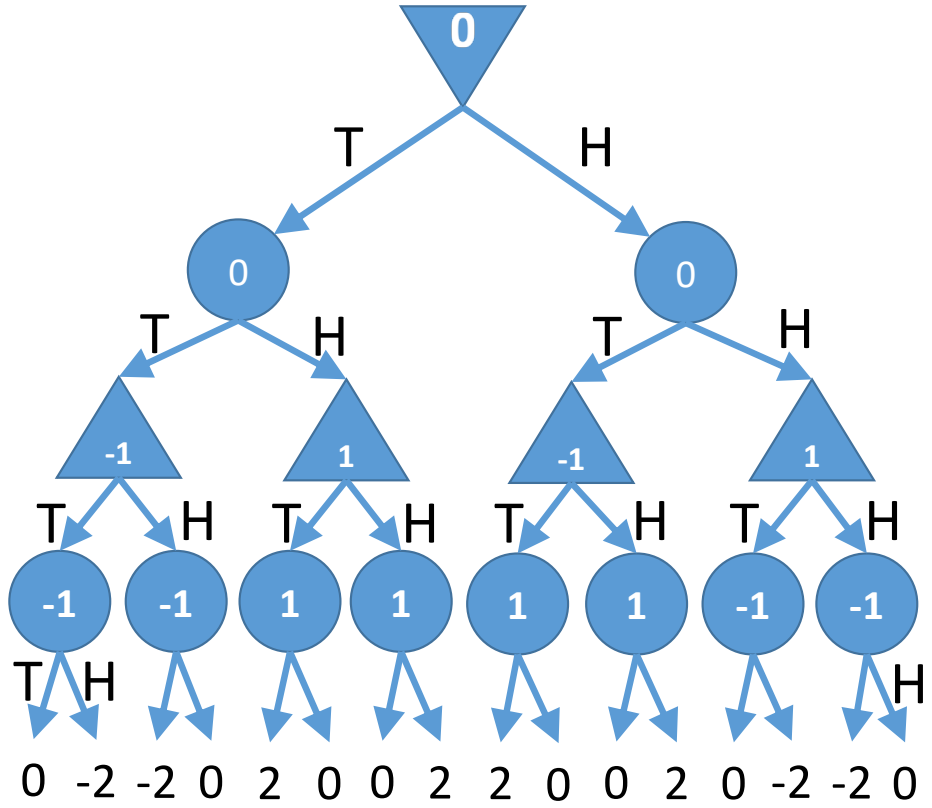
$$Q(s,a) = \sum_{s'} P(s'|s,a)U(s')$$



Expectiminimax example

Min node:

$$U(s) = \min_{a \in A(s)} Q(s, a)$$



Outline

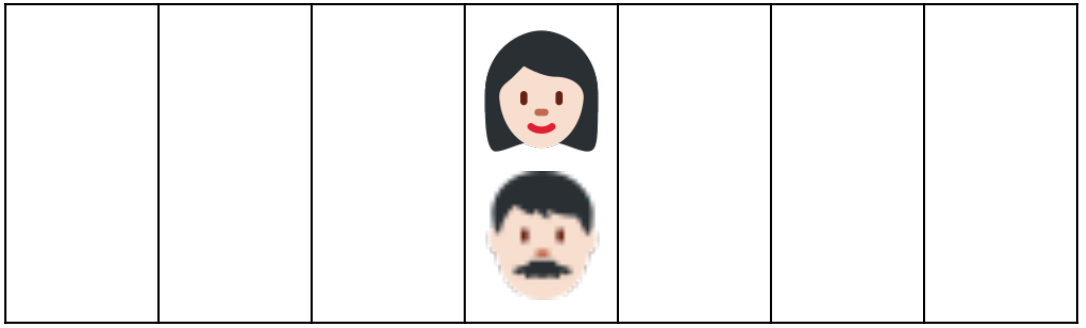
- Alpha-beta pruning
  - alpha (α) is the highest score that MAX knows how to force MIN to accept
  - beta (β) is the lowest score the MIN knows how to force MAX to accept
  - With optimum move ordering, computational complexity is $O\{n^{d/2}\}$
- Expectiminimax: Minimax search for games of chance
  - Besides MIN and MAX, there's one more player: CHANCE
  - The value of a CHANCE node is the expected value of its daughters
  - Number of levels is doubled, branching factor is large

Expectiminimax example #2

- MIN: Min decides whether she's going to move $D - 3$ or $3 - D$ steps forward, where D is the roll of the dice.
- Chance: she rolls the dice and moves her game piece in the direction indicated.
- MAX: Max decides whether he's going to move $D - 3$ or $3 - D$ steps forward, where D is the roll of the dice.
- Chance: he rolls the dice and moves his game piece in the direction indicated.

Reward: loser pays the winner a number of dollars equal to the number of spaces difference.

By Kolby Kirk, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=3037476

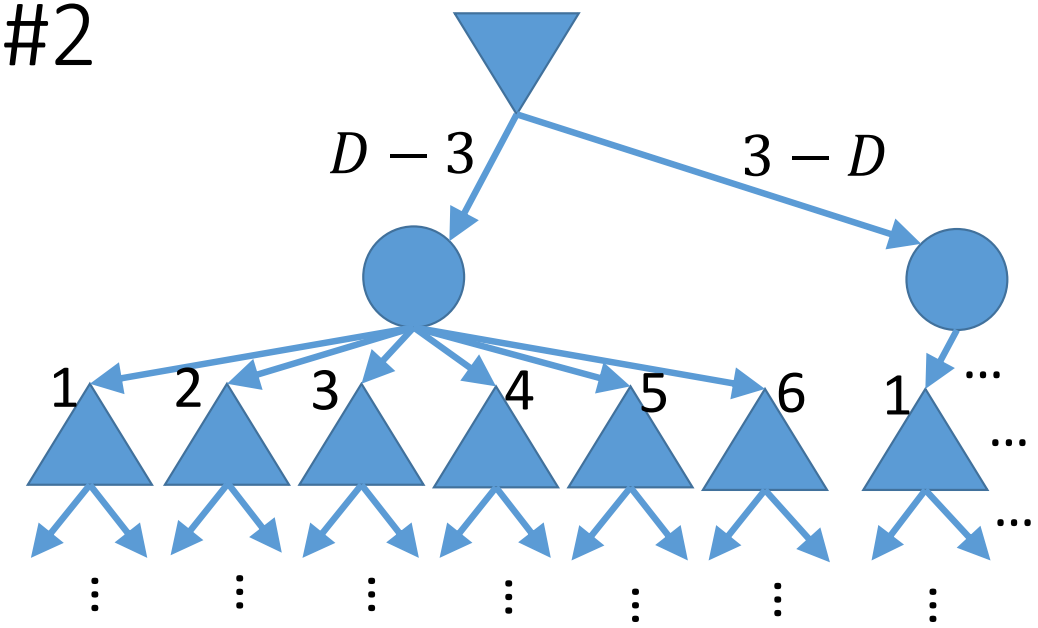


Emojis by Twitter, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=59974366.

Expectiminimax example #2

- MIN: Min decides whether she's going to move $D - 3$ or $3 - D$ steps forward, where D is the roll of the dice.
- Chance: she rolls the dice and moves her game piece in the direction indicated.
- MAX: Max decides whether he's going to move $D - 3$ or $3 - D$ steps forward, where D is the roll of the dice.
- Chance: he rolls the dice and moves his game piece in the direction indicated.

Reward: loser pays the winner a number of dollars equal to the number of spaces difference.



Outline

- Alpha-beta pruning
  - alpha (α) is the highest score that MAX knows how to force MIN to accept
  - beta (β) is the lowest score the MIN knows how to force MAX to accept
  - With optimum move ordering, computational complexity is $O\{n^{d/2}\}$
- Expectiminimax: Minimax search for games of chance
  - Besides MIN and MAX, there's one more player: CHANCE
  - The value of a CHANCE node is the expected value of its daughters
  - Number of levels is doubled, branching factor is large